DUAL SIMPLEX ALGORITHMS ON NETWORK FLOW PROBLEMS AND EXTENSIONS

By

CANAN A. SEPIL

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1987

ACKNOWLEDGEMENTS

I would like to express my deepest gratitute to Professor Suleyman Tufekçi, the chairman of my supervisory committee, for his constant and fruitful guidance and his patience.

I thank Professor Selçuk Erenguç for spending a considerable amount of time with me to discuss several topics related to my dissertation.

I would also like to thank the other members of my committee, professors Horst Hamacher, Jack Elzinga, Ralph Swain, and Chung-Yee Lee. Each has earnestly sought to enhance my education at every stage of this work.

I am grateful to my husband, Mehmet, for his support and confidence.

TABLE OF CONTENTS

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

DUAL SIMPLEX ALGORITHMS ON NETWORK FLOW PROBLEMS
AND EXTENSIONS

By

Canan A. Sepil

December 1987

Chairman: Suleyman Tufekçi
Major Department: Industrial and Systems Engineering

In this dissertation we consider special cases of optimal differential problems, the dual of the uncapacitated minimum cost network flow problem and project scheduling problem, and develop algorithms for solving these problems.

We develop three dual simplex algorithms for the minimum cost flow problem, and its special cases, the transportation problem and the assignment problem. The first algorithm provides a polynomial bound for the assignment problem. The other two algorithms use scaling techniques and provide a polynomial bound for the minimum cost flow problem and the transportation problem.

We study the project scheduling problem with the objective of maximizing the present value of the cash flows and develop an algorithm which solves this nonlinear programming problem optimally.

We present computational experience obtained by comparing the algorithms with related algorithms in the literature.

CHAPTER 1
INTRODUCTION

1.1 Purpose of the Study

There are a number of special structured mathematical programming models that are referred to as "network models." These special structures can be exploited to allow the construction of efficient algorithms. In this research, we will examine some network models that are referred to as optimal differential problems [1].

Optimal differential problems are those optimization problems in which the contraint set consists of differentials of the potentials. Rockafellar [1] discusses some special cases of linear differential problems. These special cases include the duals of transportation problems, the duals of assignment problems, and project scheduling problems. He presents a simplex algorithm for solving these problems. For the special cases, this algorithm is equivalent to the dual simplex algorithm. However, a specific rule for pivoting is not given for this simplex algorithm.

In this study, we will consider the dual simplex approach in solving the uncapacitated minimum cost flow problem with its special cases, the transportation problem and the assignment problem. Recently, dual simplex approaches in solving these problems have been studied in

the literature in the search for finding polynomially bounded algorithms. A particular leaving arc selection rule allowed us to develop dual simplex algorithms with polynomial bounds. Moreover, an algorithm that solves the project scheduling problem with the objective function of maximizing the present value of the cash flows is developed. This algorithm uses ideas similar to that of the dual simplex algorithm for the minimum cost flow problem.

1.2 Scope of the Study

In each of the remaining chapters, we will concentrate on one specific problem. Each problem will be defined, and the related literature will be discussed within the chapter. Computational experimentation comparing the discussed algorithms with related algorithms in the literature will be reported.

In Chapter 2, we will develop three polynomially bounded dual simplex algorithms for the minimum cost flow problem. In two of algorithms the scaling approach will be considered.

In Chapters 3 and 4, special cases of the minimum cost flow problem, transportation and assignment problems, will be studied.

In Chapter 5, the project scheduling problem is studied, and an algorithm which solves this nonlinear programming problem optimally is presented.

The notation used throughout the dissertation will be summarized in the Appendix.

CHAPTER 2
THE MINIMUM COST FLOW PROBLEM

The minimum cost flow problem arises whenever items must be shipped from several sources through a network in response to demand with the minimum cost of shipment. Problems of this type occur in the analyses of several systems, including production-distribution systems, urban traffic systems, railway systems, military logistic systems and communication systems.

2.1 Formulation of the Problem

Consider a network representation $G'=(V',E')$ of a minimum cost flow problem with $V'=V_S' \cup V_D'$, where V_S' is the set of supply nodes and V_D' is the set of demand nodes. If there exists a direct arc from $i \in V'$ to $j \in V'$, then $(i,j) \in E'$. For each arc $(i,j) \in E'$, we associate an integer c_{ij} which corresponds to a unit shipping cost over the arc (i,j). Moreover, with each node $i \in V_S'$, we associate a positive integer a_i designating the supply at that point and with each node $j \in V_D'$, we associate a nonnegative integer b_j corresponding to the demand at point j. If $b_j = 0$, then node j is called a transshipment node.

For convenience, for each node $i \in V'$, let $\alpha(i)$ represent the "forward star" of node i which is the set of all the outgoing arcs from

node i, and let $\beta(i)$ represent the "backward star" of node i which is the set of all the incoming arcs to node i.

The minimum cost flow problem (MCP) can now be stated as a linear programming problem :

$$\text{(MCP)} \quad \text{Minimize} \sum_{(i,j)\in E'} c_{ij}x_{ij} \tag{2.1}$$

subject to

$$\sum_{(i,j)\in\alpha(i)} x_{ij} - \sum_{(i,j)\in\beta(i)} x_{ij} = \begin{cases} a_i & \text{if } i\in V_S' \\ -b_i & \text{if } i\in V_D' \end{cases} \tag{2.2}$$

$$x_{ij} \geq 0 \qquad (i,j)\in E' . \tag{2.3}$$

In what follows we assume that the sum of supplies is equal to the sum of demands, i.e.,

$$\sum_{i\in V_S'} a_i = \sum_{j\in V_D'} b_j . \tag{2.4}$$

In Section 2.3, we present the dual of (MCP). In Section 2.4, we provide some definitions on the tree structure of a dual feasible solution. Also in this section, we present the idea of a dual simplex pivot, the "induced subtree" of a given node and a general overview of the algorithm and present the algorithm for solving the minimum cost flow problem. Section 2.4 also presents the validity and the computational bound of the algorithm. In Sections 2.5 and 2.6, two scaling algorithms with polynomial bounds are developed for the minimum cost flow problem.

2.2 Literature Review

The transportation problem, a special case of minimum cost flow problem, was first solved by Hitchcock [2] in 1941 and Koopmans [3] in 1946. In 1951, Dantzig [4] developed a variant of the simplex algorithm for the transportation problem. Orden [5] extended these results for the minimum cost flow problem.

Early approaches other than the primal simplex that have been proposed are out-of-kilter by Fulkerson [6], primal-dual by Ford and Fulkerson [7], dual by Balas and Hammer [8], flow augmentation by Busacker and Gowen [9], and negative cycle method by Klein [10].

In 1950s and early 1960s, the out-of-kilter algorithm was seen as superior to all the other methods. Development of fast and efficient codes for the primal problem begun in 1970's with the development of new computer science tools. Currently, based on the experiments which compare the different algorithms for minimum cost flow problems, it is believed that primal simplex algorithms are superior to all the others [11,12] since primal implementations are faster and require less storage.

The successful implementations of the primal simplex network algorithms are based on compact representation of the basis, efficient ways of determining the entering and leaving arcs and efficient techniques to update the simplex multipliers at each pivot.

Different data structures have been used to represent the basis. In addition to predecessor and thread labels, the following labels are proposed; "number of nodes" label by Glover and Klingman [13], "augmented predecessor" label by Glover , Karney and Klingman [14],

"augmented thread" label by Glover, Klingman and Stutz [15], "preorder distance" label by Bradley, Brown and Graves [16]. Ali et al.[17] conclude that data structures which use the thread and predecessor labels in conjunction with other labels are the most efficient data structures available for simplex specializations.

Efficient methods to perform pricing for determining the entering arc generally include a candidate queue [16,18] which is used to price only a subset of the nonbasic variables at each pivot.

Even though primal simplex algorithms are superior to other methods computationally, they may fail to terminate due to an infinite number of degenerate pivots caused by cycling. In order to prevent cycling, Cunningham [19], and Barr, Glover and Klingman [20] proposed algorithms which work only with strongly feasible trees. A strongly feasible tree is defined to be a feasible tree in which every arc with a flow of zero is directed away from the root. Cycling is prevented by a specific choice of the leaving arc. Bland's [21] rule for choosing the entering and leaving arcs (choosing the variable with the least index possible) also prevents cycling.

In the absence of cycling, primal simplex algorithms can still encounter an exponentially long sequence of consecutive degenerate pivots. This phenomenon is called stalling. In order to avoid stalling, Cunningham [22] proposed to use some entering-arc rules combined with the maintenance of strongly feasible trees. For this, each arc is considered periodically as a choice for entering arc. The rules that maintain these are Least Recently Considered, Least Recently Basic, and

Inward Most Negative rules. Rules similar to these are also suggested in other studies [12,23,24].

However, Zadeh [25] proved that when Dantzig's rule for the entering variable is used, the primal simplex algorithm may take an exponential number of pivots on some problems while all the bases remain nondegenerate. Also he presented bad networks for primal-dual, path and cycle methods.

The algorithms that are of primal-dual type include the following:

Tomizawa [26] developed a variant of the ordinary primal-dual algorithm in which the associated shortest path problem was solved by Dijkstra's algorithm.

Bertsekas [27] introduced a unified framework for primal-dual methods in minimum cost network flow problems by giving different procedures for implementing flow augmentations, price adjustments and ascent steps.

Following Zadeh's findings, algorithms other than primal simplex and primal-dual have been considered in the hopes of finding polynomially bounded algorithms.

Edmonds and Karp [28] were the first to solve minimum cost network flow problems in polynomial time. Their algorithm uses a scaling technique and solves a sequence of $O[\tau]$ different network flow problems where $2^{\tau} < c(u,v)$ for all arcs (u,v) and c(.,.) is the capacity of an arc. These problems differ in their arc capacities where

for problem p, capacities are obtained by deleting the p low-order digits in the binary representations of the original capacities. The algorithm has a computational bound of $O(\tau|V|^4)$. Orlin [29] proved that a variant of the Edmonds-Karp scaling algorithm solves the minimum cost network flow problem in $O((|V|^2\log|V|)(|E|+|V|\log|V|)$, a bound independent of input data.

Röck [30] described a variant of the Edmonds and Karp scaling algorithm with a computational bound of $O(\alpha|V|^4)$ where $2^{\alpha} < C(u,v)$ for all arcs (u,v) and C(.,.) is the cost of an arc. Here the bound depends on the size of the costs.

Orlin [29] also developed two genuinely polynomial dual simplex algorithms for the minimum cost flow problems that have bounds of $O[U^*(|V|\log |V| + |E|)]$ and $O[U^*(|V|)]$ pivots where $U^* = \min [|V|^2 \log |V|, |V|BIT(b)]$. A genuinely polynomial algorithm is an algorithm that has a computational bound which is polynomial in the number of nodes and is independent of both costs and capacities. Here BIT(b) is the minimum integral value of s such that $2^s b$ is integer-valued. In Orlin's formulation of the problem, the vector b is the vector of weights of the nodes and it is scaled so that $|b_i| \leq 1$. The computational bounds of these algorithms are $O(U^*|V|^4)$ and $O(U^*|V|^2)$ respectively. Orlin's algorithms use the idea of the Edmonds and Karp scaling method.

Armstrong, Klingman and Whitman [31] introduced a variant of the dual simplex algorithm in which different list structures for storing

the basis were considered and multiple pivot strategy was employed. No computational bounds were given for their algorithm.

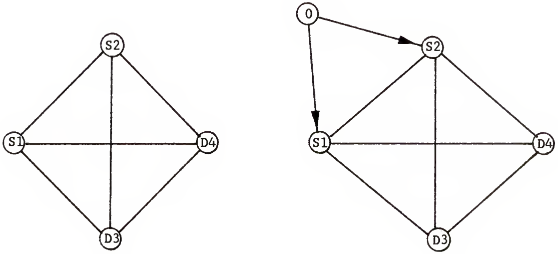
Ikura and Nemhauser [32,33] provided a dual simplex algorithm for the transportation problem with an upper bound on the simplex pivots polynomial in the length of the input data. More precisely, the algorithm has a bound of $O(|V|^2M)$ simplex pivots where M is the sum of the supply and demand. Moreover, if the Edmonds and Karp algorithm is used in a recursive manner to adjust the supplies and demands, the upper bound on the number of simplex pivots of the revised algorithm becomes a polynomial of $O(|V|^3(\tau+1))$ where τ is defined as $2^\tau \le \max w_j \le 2^{\tau+1}$. Here w_j is the weight (supply or demand) of node j. This algorithm has a computational bound of $O(|V|^5(\tau+1))$.

Recently, Tardos [34] develeloped a genuinely polynomial algorithm for the minimum-cost circulation problem. Her algorithm has a computational bound of $O(|E|^2T(|E|,|V|)\log |E|)$. Here $T(|E|,|V|)$ is the order of solving a maximum flow problem in a network with $|E|$ arcs and $|V|$ nodes. Her algorithm solves at most $|E|$ subproblems obtained by scaling the cost coefficients.

Fujishige [35] provided an algorithm for the dual of Tardos's problem which runs in $O(|E|(|V|+m_0)^2\log |E|)$ time where m_0 is the number of arcs with both finite lower and upper capacities. His algorithm solves at most $|E|$ subproblems obtained by scaling the capacities.

2.3. The Dual Formulation

Consider the augmented graph $G=(V,E)$ which is obtained from $G'=(V',E')$ by appending a "super source" with node number 0 connected to each source node $j \in V_S'$ via an arc $(0,j)$ with associated costs $c_{0j} = 0$. Let $a_0 = 0$. This situation is depicted in Figure 2.1 below. The types of nodes, whether a source node or a demand node, are indicated by the letters S or D before the node numbers.



$G'=(V',E')$ $\qquad\qquad$ $G=(V,E)$

Figure 2.1: The graphs G' and G

Consider the dual of the LP formulation of (MCP) by using G.

(DMCP) $$\text{Max} \sum_{i \in V_S} a_i w_i - \sum_{j \in V_D} b_j w_j \tag{2.5}$$

subject to

$$w_i - w_j \le c_{ij} \qquad (i,j) \in E. \tag{2.6}$$

2.4. A Dual Simplex Algorithm for the Minimum Cost Flow Problem

In this section, we will first discuss separately the main steps of the algorithm; initialization, determination of the leaving and entering arcs. Then, we will present the algorithm in Section 2.4.6.

2.4.1 Notation

Throughout Section 2.4 we will use the following notation.

A path in G is an alternating sequence $P_k = v_0, e_1, \ldots, e_k, v_k$ of nodes and arcs such that $e_i = (v_{i-1}, v_i)$ or $e_i = (v_i, v_{i-1})$. In the former case, e_i is called a forward arc of P. In the latter case it is called a backward arc of P.

Let $T=(V,E_T)$ be a spanning tree rooted at node 0. Define PR(x) to be the predecessor function indicating the node which precedes node x on the unique path from node $x \in V$ to node 0 in T. Let D(i) represent a depth function defined as the number of nodes traversed from node 0 to node i on a unique path in T.

Let V_F be the set of nodes $i \in V$ such that the arc (PR(i),i) is a forward arc in T. Similarly, let V_B be the set of nodes $i \in V$ such that the arc (PR(i),i) is a backward arc in T.

2.4.2 Construction of the Initial Tree

Let $T_o = (V,E_T)$ be a shortest path tree of G rooted at node 0, using costs as the weights. If all the costs are nonnegative, we can solve the shortest path problem using Dijkstra's algorithm in $O(|V|^2)$

time. If the costs are not nonnegative, then we can solve the shortest path problem using the label-correcting algorithm for shortest paths in $O(|V|^3)$ time. In this study, we assumed that there exists no negative cycles in G. Once we solved the shortest path problem, we let w_i represent the negative of the shortest path from node 0 to node i in T_0.

Proposition 1: W_i, $i \in V$, represents a basic feasible solution for (DMCP), the dual problem.

Lemma 2.1: If the tree T_0 possesses a feasible solution for the (MCP) by only allowing the tree arcs to permit flow, then the solution is optimal for the (MCP).

Proof: Primal feasibility, dual feasibility and complementary slackness are all satisfied.

2.4.3 Checking Primal Feasibility

Let $T=(V,E_T)$ be a spanning tree of G which possesses a dual feasible solution. We call a tree $T_i=(V_i,E_i)$ the subtree "induced" by node $i \in V_F$ if this subtree is obtained by dropping arc (PR(i),i) and taking the subtree that does not contain the root node. (See Figure 2.2)

Let $V_i = V_{iS} \cup V_{iD}$ where V_{iS} is the set of supply nodes in T_i and V_{iD} is the set of demand nodes in T_i. Let

$$S_i = \sum_{j \in V_{iS}} a_j - \sum_{j \in V_{iD}} b_j. \tag{2.7}$$

<u>Lemma 2.2</u>: Given a rooted spanning tree $T=(V,E_T)$ which possesses a dual feasible solution, if there exists a node $i \in V_F$ such that $S_i>0$, then the corresponding primal solution is infeasible.

<u>Proof</u>: Since $S_i>0$, if arc $(PR(i),i)$ is the dropped arc in obtaining the induced subtree T_i, then in the computation of flows, by only using the tree arcs, we will have $x_{(PR(i),i)} = -S_i < 0$.

2.4.4 Determination of the Leaving Arc

Given an induced subtree $T_i = (V_i,E_i)$, we define a subtree $T_i' = (V_i',E_i')$ of T_i and its corresponding S_i' value by the procedure PRUNE1. Before stating the exact steps, we will give the general idea of the procedure. Initially, given an induced subtree T_i, we set S_j' to be equal to S_j for all nodes $j \in T_i \cap V_F$. Then we start from the tip nodes in T_i, and eliminate all subtrees T_j with positive S_j' values from T_i. Once we eliminate a subtree, we update the S_i' value in the remaining subtree. When all the subtrees with positive updated S_j' values are eliminated from T_i, the resulting tree is called T_i' with its associated S_i' value.

PROCEDURE "PRUNE1"

INPUT : $T_i = (V_i,E_i)$, $D(k)$, $k \in V_i \cap V_F$.

OUTPUT : A subtree $T_i' = (V_i',E_i') \subseteq T_i$ with the property that $S_j' \le 0$ for each $j \in V_{iF} \cap V_i'$ and its S_i' value.

BEGIN

INITIALIZE : Set $T_j' \leftarrow T_j$, and $S_j' = S_j$ for all $j \in V_i \cap V_F$.

Let $R = \{ j \neq i \in V_i \cap V_F : S_j' > 0 \}$.

$Z_i = \emptyset$.

WHILE $R \neq \phi$ DO

Find $j \in R$ such that $D(j) \geq D(k)$, for all $k \neq j \in R$. Obtain $T_j' = (V_j', E_j')$.

Set $T_i' = (V_i' \backslash V_j', \; E_i' \backslash E_j' \cup \{(PR(j), j)\})$.

$Z_i \leftarrow Z_i \cup \{j\}$.

Update S_j', $j \in V_i' \cap V_F$ and R.

ENDWHILE

Compute $S_i' = \sum_{j \in V_{iS}'} a_j - \sum_{j \in V_{iD}'} b_j$ (2.8)

where V_{iS}' and V_{iD}' are the sets of supply and demand nodes in T_i' respectively.

END

Note that $Z_i = \{j : j \in V_F \cap V_i, \; j \neq i, \; S_j' > 0\}$ is the set of nodes whose subtrees are pruned from subtree T_i with the procedure PRUNE1.

We will also define a set Z_i' where

$$Z_i' = \begin{cases} Z_i \cup \{i\} & \text{if } S_i' > 0 \\ Z_i & \text{otherwise.} \end{cases}$$

Note that for a node $i \in V_F$, S_i is the negative of the flow on arc $(PR(i), i)$ which defines the induced subtree T_i; and S_i' is the negative of the flow on arc $(PR(i), i)$ when T_i' is used for computation of flows.

This might be viewed as the individual contribution of the subtree T_i' to the flow on arc (PR(i),i).

<u>Example 2.1.</u> Consider the minimum cost flow problem in Figure 2.2. It is given that $a_1 = 10$, $a_2 = 20$, $a_3 = 20$, $a_4 = 10$, and $b_5 = 5$, $b_6 = 3$, $b_7 = 1$, $b_8 = 1$, $b_9 = 0$, $b_{10} = 7$, and $b_{11} = 43$. Then, $S_{10} = 33$, $S_{10}' = 14$, $S_8 = 19$ and $S_8'=19$.

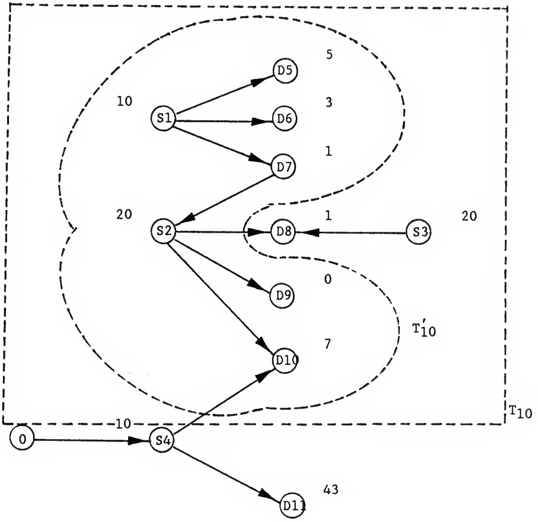


Figure 2.2. A Spanning Tree for Example 2.1

In the algorithm, the leaving arc is obtained in the following manner:

Let Q be the set of nodes $j \in V_F$ with $S_j' > 0$. A subtree T_p, $p \in Q$, will be selected for pivot in such a way that no other node $j \in V_F$ on the path from node p to node 0 will have $S_j' > 0$. Initially, an easy way of determining T_p is to check the depth functions of the nodes in Q and select T_p such that $D(p) = \min\{D(j): j \in Q\}$. Throughout the algorithm, after a particular pivot k, T_p is determined while updating T^{k+1}. Let $P_{p'}$ be the set of nodes on the unique path from node p found in pivot k, p', to node 0 in T^{k+1}. Then new p is chosen to be the node $\alpha \in P_{p'} \cap Q$ with the smallest depth function. If $P_{p'} \cap Q = \phi$, then p is chosen to be the node $\alpha \in Q$ with the smallest depth function, and p' is set to p. Once a subtree for pivoting is determined, the leaving arc is then the dropped arc ((PR(p),p) in determining the induced subtree T_p.

2.4.5. Determination of the Entering Arc

From the construction of the dual problem, we have

$$w_i - w_j \le c_{ij} \qquad (i,j) \in E. \tag{2.9}$$

Moreover,

$$w_i - w_j = c_{ij} \qquad \text{if } (i,j) \in E_T. \tag{2.10}$$

Let T_p be the subtree selected for pivoting. Also let

$$\Delta = \min_{\substack{i \in V_p \\ j \in V \setminus V_p}} \{ \Delta_{ij} = w_j - w_i + c_{ij} \} = \Delta_{fl} \ge 0 \quad (f,l) \in E \setminus E_T. \tag{2.11}$$

Then we enter arc (f,l) into the tree T and drop arc (PR(p),p) out of the tree and update the dual variables as follows :

$$w_i = w_i + \Delta \qquad \text{if } i \in V_p \tag{2.12}$$

$$w_i = w_i \qquad \text{if } i \in V \backslash V_p. \tag{2.13}$$

Lemma 2.3: This dual solution is a basic feasible solution which can be characterized by its associated rooted tree T' obtained from T by deleting arc (PR(p),p) and entering arc (f,1).

Proof: We note that for all arcs (z,y) with $z \in V_p$, $y \in V_p$ or $z \in V \backslash V_p$, $y \in V \backslash V_p$, the inequality does not change. Moreover for an arc (z,y), $z \in V_p$, $y \in V \backslash V_p$, the inequality holds due to the selection of Δ. Finally for the arcs (z,y), $z \in V \backslash V_p$, $y \in V_p$ we have

$$w_z - (w_y + \Delta) \leq c_{ij} \tag{2.14}$$

which is also satisfied since $\Delta \geq 0$.

Each time such a tree change (a dual simplex pivot) is performed a new arc is added to the tree which will try to funnel the excess supply in T_p to a new node $1 \notin V_p$. Moreover, since the arc (PR(p),p) is dropped out of T, it will have zero flow in the current attempt to find the primal feasible solution.

The algorithm to be presented in the next section will perform the dual simplex pivots as follows : First a node $p \in Q$, with the property that for all other nodes $j \in V_F$ on the unique path from node p to node 0 have $S_j' \leq 0$, will be selected. Then the arc (PR(p),p) will be dropped out of T and an arc $(f,1) \notin E_T$ will enter the tree T. Here (f,1) is an out-of-tree arc which yields the value Δ calculated in (2.11). This process will be repeated until $S_i' = S_i \leq 0$ for all $i \in V_F$. At that point $Q = \phi$, and the current spanning tree possesses an optimal solution to the primal problem. The validity of this algorithm and optimality results are presented in Section 2.4.7.

2.4.6 The Dual Simplex Algorithm, DUAL

In this section, we present the general algorithm, DUAL, for solving the (MCP).

Step 0.(Initialization) Construct the shortest path tree of G rooted at node 0 by using costs as weights. Let w_i, $i \in V$ represent the negatives of the shortest path lengths form node 0 to node i. Let $T=(V,E_T)$ be this tree with dual variables, w_i, $i \in V$. Compute the depth function D(i), and S_i' by using procedure PRUNE1 , $i \in V_F$.

Let $Q = \{i: i \in V_F, S_i' > 0\}$.

Set p' <--- 0.

Set k <--- 0.

Step 1.WHILE $Q \neq \phi$ DO

BEGIN

k <--- k+1

Step 1a. (Determination of the leaving variable)

IF $p'=0$ THEN

Find T_p such that $D(p) = \min \{D(j): j \in Q\}$

ELSE

Find T_p such that $D(p) = \min \{D(j): j \in Q \cap P_{p'}\}$

where $P_{p'}$ is the set of nodes on the unique path from node p' found in iteration k-1 to node 0 in T^k.

ENDIF

Let (PR(p),p) be the arc which is dropped to obtain T_p.

Step 1b. (Determination of the entering variable)

Find $\Delta = \min \{\Delta_{ij} = w_j - w_i + c_{ij} : i \in V_p,\ j \in V \setminus V_p,\ (i,j) \in E\} = \Delta_{f1}$

Let $w_j \leftarrow w_j + \Delta \quad j \in V_p$

$w_j \leftarrow w_j \quad j \in V \setminus V_p$

END

Step 1c. (Updating)

Update T, S_i, S_i', Q.

IF $P_p \cap Q = \phi$, THEN

Set $p' = 0$

ELSE

Set $p' = p$.

ENDWHILE

The current tree possesses an optimal solution to (MCP).

END

Example 2.2. The minimum cost flow problem with three source nodes and four demand nodes is solved by the algorithm DUAL in Figure 2.3.

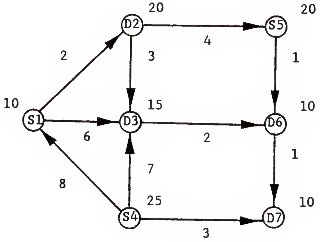
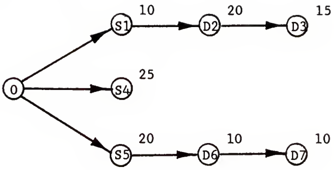


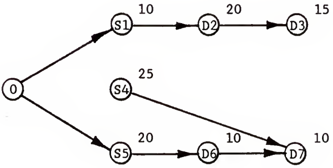
Figure 2.3 An Example

Initialization



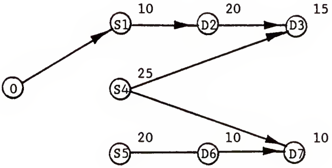
i	w_i
1	0
2	-2
3	-5
4	0
5	0
6	-1
7	-2

Iteration 1. $S_4' = 25$, $Q = \{4\}$, $p=4$. Arc (0,4) leaves, arc (4,7) enters, $\Delta=1$.



i	w_i
1	0
2	-2
3	-5
4	1
5	0
6	-1
7	-2

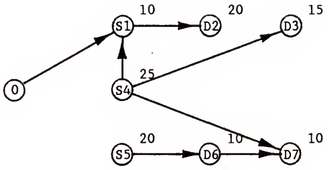
Iteration 2. $S_7'=15$, $S_5'=10$. $Q=\{5,7\}$, $p=5$. Arc (0,5) leaves, arc (4,3) enters, $\Delta=1$.



i	w_i
1	0
2	-2
3	-5
4	2
5	1
6	0
7	-1

Figure 2.3 (continued)

Iteration 3. $S_3' = 10$. $Q = \{3\}$, $p=3$. Arc (2,3) leaves, arc (4,1) enters, $\Delta=6$.



i	w_i
1	0
2	-2
3	1
4	8
5	7
6	6
7	5

Iteration 4. $Q = \phi$. We have the optimum solution.

Figure 2.3. (continued)

2.4.7 The Validity of the Algorithm

The validity of the algorithm is based on the fact that throughout the algorithm, the flows on the backward arcs of T remain positive.

For each node $i \in V_B$, let $T_i'' = (V_i'', E_i'')$ be the subtree obtained from T by dropping the arc (i,PR(i)) and taking the subtree which contains node i.

Let $$S_i'' = \sum_{j \in V_{iS}''} a_j - \sum_{j \in V_{iD}''} b_j \qquad (2.15)$$

where V_{iS}'' and V_{iD}'' are the sets of supply and demand nodes in T_i'', respectively. Note that for a node $i \in V_B$, S_i'' is the flow on the backward arc (i,PR(i)).

Given an subtree $T_i'' = (V_i'', E_i'')$, we define a subtree $T_i''' = (V_i''', E_i''')$ of T_i'' by the procedure PRUNE2. This procedure is

similar to procedure PRUNE1 in that we are pruning subtrees with positive S_j' values. However in PRUNE2, we use the subtree T_i'' as input. In PRUNE2, given the subtree T_i'', we start from the outermost nodes of T_i'' and eliminate the subtrees T_j' with positive S_j' values and update the S_j'' values in the remainder of the subtree. Once we eliminate all subtrees with positive S_j' values, the remainder subtree is called T_i''' with its S_i''' value.

PROCEDURE "PRUNE2"

INPUT : $T_i'' = (V_i'', E_i''), i \in V_B$, $D(k)$, $k \in V_i'' \cap V_F$.

OUTPUT : A subtree $T_i''' = (V_i''', E_i''') \subseteq T_i''$ with the property that $S_j' \leq 0$ for each $j \in V_F \cap V_i'''$ and its S_i''' value.

BEGIN

INITIALIZE : Set $T_i''' \leftarrow T_i''$, $R = \{ j \in V_i'' \cap V_F : S_j' > 0 \}$.

$Z_i = \phi$.

WHILE $R \neq \phi$ DO

Find $j \in R$ such that $D(j) \geq D(k)$, for all $k \neq j \in R$. Obtain $T_j' = (V_j', E_j')$. Set $T_i''' = (V_i''' \backslash V_j', E_i''' \backslash E_j' \cup \{(PR(j), j)\})$.

$Z_i \leftarrow Z_i \cup \{j\}$.

Update S_j''', $j \in V_i''' \cap V_B$.

ENDWHILE

Compute $$S_i''' = \sum_{j \in V_{iS}'''} a_j - \sum_{j \in V_{iD}'''} b_j \qquad (2.16)$$

where V_{iS}''' and V_{iD}''' are the sets of supply and demand nodes in T_i''' respectively.

END

Here $Z_i = \{j : j \in V_F \cap V_i'', S_j' > 0\}$ is the set of nodes whose subtrees are pruned from subtree T_i'' with the procedure PRUNE2.

Example 2.3. Consider the minimum cost flow problem in Figure 2.4. It is given that $a_1 = 10$, $a_2 = 20$, $a_3 = 20$, $a_4 = 10$, and $b_5 = 5$, $b_6 = 3$, $b_7 = 1$, $b_8 = 1$, $b_9 = 0$, $b_{10} = 7$, and $b_{11} = 43$. Then $S_2'' = 40$, and $S_2''' = 21$.

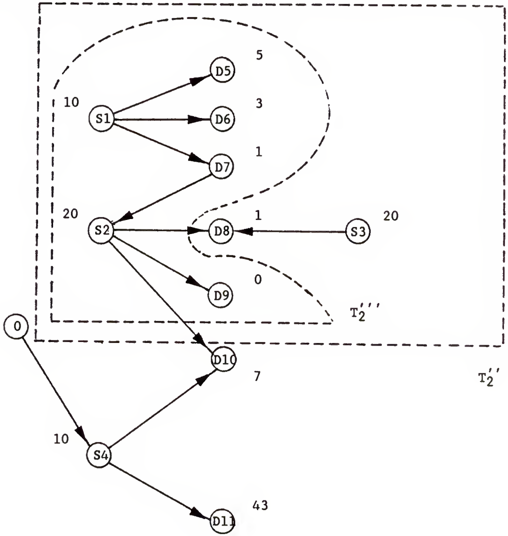


Figure 2.4. A Spanning Tree for Example 2.3

At this point, we will give an overall review of the interpretation of the subtrees T_i, T_i', T_i'' and T_i''' with their corresponding S_i, S_i', S_i'', S_i''' values and their relationships. First we note that T_i, T_i', S_i, S_i' are defined for all nodes $i \in V_F$, and T_i'', T_i''', S_i'', and S_i''' are defined for all nodes $i \in V_B$.

T_i is the subtree induced by the nodes $i \in V_F$. S_i value is the negative of the flow on the forward arc (PR(i),i).

$T_i' \subseteq T_i$ is the subtree induced by the nodes $i \in V_F$ when all the subtrees $T_j' \subseteq T_i$ with positive S_j' values are pruned from T_i. S_i' value is the negative of the flow on the forward arc (PR(i),i) in T_i'.

T_i'' is the subtree induced by the nodes $i \in V_B$. S_i'' value is the flow on the backward arc (i,PR(i)).

$T_i''' \subseteq T_i''$ is the subtree induced by the nodes $i \in V_B$ when all the subtrees $T_j' \subseteq T_i''$ with positive S_j' values are pruned from T_i''. S_i''' value is the flow on the backward arc (i,PR(i)) in T_i'''.

Thus, we can write

$$S_i' = S_i - \sum_{j \in Z_i} S_j' \quad \text{for } i \in V_F \text{ and} \tag{2.17}$$

$$S_i''' = S_i'' - \sum_{j \in Z_i} S_j' \quad \text{for } i \in V_B. \tag{2.18}$$

where the set Z_i is the set of nodes whose subtrees are pruned from subtree T_i or T_i'' using procedures PRUNE1 or PRUNE2.

The update of the S_i, $i \in V_F$, and S_i'', $i \in V_B$, values after a particular pivot is nothing but flow update as in the primal simplex method around the unique cycle formed by introducing the entering arc into the spanning tree. That is, we add the S_p value to all the flows on the arcs that have the same orientation with the cycle, and subtract S_p

value from the flows on the arcs that have the opposite orientation with the cycle.

We will use superscripts $(.)^b$ and $(.)^a$ to indicate a subtree (.) or a value (.) before and after a particular pivot, respectively.

Let a pivot involving T_p bring an arc $(f,l) \notin E_T$, $f \in T_p$, $l \notin T_p$ into the tree. Let (x,p) be the leaving arc and let θ be the first common node (join) between P_p^b and $P_l{}^b$. Consider the unique cycle formed by introducing the arc (f,l) into T^b (Figure 2.5). We can divide the cycle into three separate paths and examine the updating of S_i, $i \in V_F{}^b$ and S_i'' $i \in V_B{}^b$ values for nodes in each path separately.

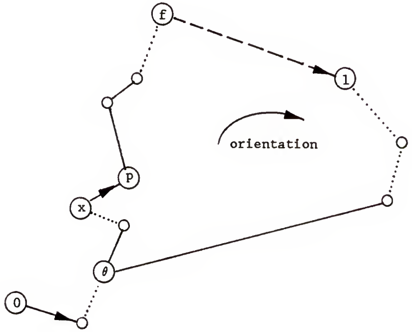


Figure 2.5. The Unique Cycle Formed by Introducing Arc (f,l) into T

For nodes $i \in P_x^b \backslash P_\theta{}^b$, if node $i \in V_F{}^b$, then $i \in V_F{}^a$. Similarly, if node $i \in V_B{}^b$, then $i \in V_B{}^a$. That is the orientation of the arcs on this path do not change after the pivot. Thus, we update the values as

$$- S_i{}^a = - S_i{}^b + S_p{}^b \qquad i \in V_F{}^a, \tag{2.19}$$

$$S_i''{}^a = S_i''{}^b - S_p^b \qquad i \in V_B{}^a. \tag{2.20}$$

Note that the negative signs before S_i values in Equation (2.19) is from the fact that S_i value for a node $i \in V_F$ is the negative of the flow on the arc (PR(i),i).

For nodes $i \in P_1{}^b \backslash P_\theta{}^b$, if node $i \in V_F{}^b$, then $i \in V_F{}^a$. Similarly, if node $i \in V_B{}^b$, then $i \in V_B{}^a$. Again, the orientation of the arcs on this path do not change after the pivot. Thus, we update the values as

$$- S_i{}^a = -S_i{}^b - S_p^b \qquad i \in V_F{}^a, \tag{2.21}$$

$$S_i''{}^a = S_i''{}^b + S_p^b \qquad i \in V_B{}^a. \tag{2.22}$$

Note that for nodes $i \in P_f{}^b \backslash P_p^b$, if $i \in V_F{}^b$ then $i \in V_B{}^a$. Similarly, if $i \in V_B{}^b$, then $i \in V_F{}^a$. That is the orientation of the arcs on this path changes after the pivot. Let $v_i = PR(i)$ in T^b. Then we update the values as

$$S_{v_i}^a = -(S_i''{}^b - S_p^b) \qquad v_i \in V_F{}^a, \tag{2.23}$$

$$S_{v_i}''{}^a = -S_i{}^b + S_p^b \qquad v_i \in V_B{}^a. \tag{2.24}$$

Also for the node f, $f \in V_B{}^a$, we have

$$S_f''{}^a = S_p^b. \tag{2.25}$$

Lemma 2.4. Let T_p be the subtree that is selected to be pivoted on. Then $S_p > 0$. Moreover $S_p > S_k$, $k \in V_p \cap V_F$, $k \neq p$.

Proof: From Equation (2.17),

$$S_p = S_p' + \sum_{k \in Z_p} S_k' = \sum_{k \in Z_p'} S_k'. \tag{2.26}$$

Since $p \in Q$, $S_p' > 0$, and $\sum_{k \in Z_p} S_k' > 0$, it follows that $S_p > 0$.

Moreover, for any $k \neq p \in V_p \cap V_F$, we have

$$S_k = S_k' + \sum_{i \in Z_k} S_i'. \tag{2.27}$$

If $S_k' > 0$, then since $Z_k \cup \{k\} \subseteq Z_p$, it follows that $S_p > S_k$. On the other hand, if $S_k' \leq 0$, then $Z_k \subseteq Z_p$ and thus $S_p > S_k$. ∎

The following lemma states that the flows on all the backward arcs remain positive throughout the algorithm.

Lemma 2.5. Throughout the algorithm, the following conditions remain true.

i) $S_i'' > 0$ for all nodes $i \in V_B$,

ii) $S_i''' > 0$ for all nodes $i \in V_B$.

Proof: It is obvious that the statement is true for T^0 since $V_B = \phi$ in T^0. Assume it is true at the end of iteration k. At iteration (k+1) let a pivot involving T_p bring an arc $(f,l) \notin E_T$, $f \in T_p$, $l \notin T_p$ into the tree.

Since the S_i'' values of only the nodes in the unique cycle shown in Figure 2.5 are changed by the pivot, we will prove the conditions remain true for only those nodes.

a) First we will consider the nodes $i \in (P_x^b \backslash P_\theta{}^b) \cap V_B{}^b$. (Figure 2.6) From the definition of the procedure PRUNE2, it is obvious that

$$S_i'''^a = S_i'''^b \tag{2.28}$$

since $S_p' > 0$ and thus T_p is discarded from T_i''' in the calculation of $S_i'''^b$.

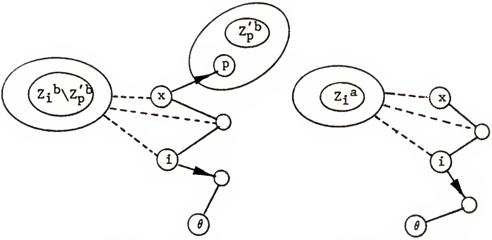
Using Equations (2.18), (2.28) and the induction hypothesis, we have

$$s_i'''^{b} = s_i'''^{a} = s_i''^{a} - \sum_{j\in Z_i^a} s_j'^{a} > 0 \quad . \tag{2.29}$$

Thus,

$$s_i''^{a} > \sum_{j\in Z_i^a} s_j'^{a} \ge 0, \tag{2.30}$$

which completes the proof for case a).



Before the pivot After the pivot

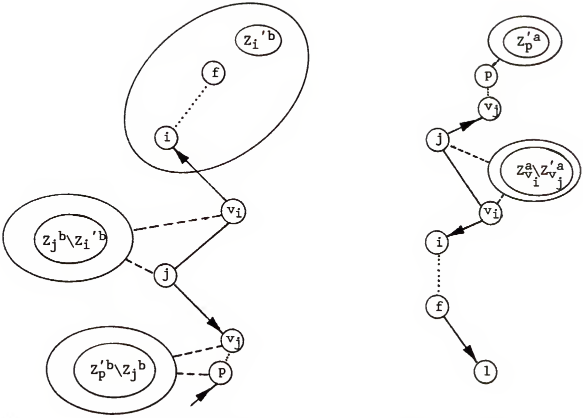
Figure 2.6 Nodes between x and θ before and after the Pivot

b) Consider the nodes $i\in(P_f^b \backslash P_p^b)\cap V_F^b$. (Figure 2.7)

Letting v_i = PR(i) ,$i\in(P_f^b \backslash P_p^b)\cap V_F^b$ in T^b, using Equation (2.24), and noting that $v_i \in V_B^a$, we have

$$s_{v_i}''^{a} = -s_i^b + s_p^b \tag{2.31}$$

From Lemma 2.4, $s_p^b > s_k^b$, $k\in V_F^b \cap V_p^b$, thus $s_{v_i}''^{a} > 0$ for $v_i \in V_B^a$.



Before the pivot After the pivot

Figure 2.7 Nodes between f and p before and after the Pivot

To show that condition ii) remains true on this path after the pivot consider a node $i \in V_F{}^b$ and a node $j \in V_B{}^b$ such that $i,j \in P_f{}^b \backslash P_p^b$ with $D(j)<D(i)$, and $S_{v_j}'^a > 0$. (Figure 2.7) Let v_i=PR(i) and v_j=PR(j) in T^b. From Equation (2.18), by dividing the set $Z_j{}^b$ into two disjoint subsets, and by the induction hypothesis, we have

$$S_j'''^b = S_j''^b - \sum_{k \in Z_j{}^b \backslash Z_i'^b} S_k'^b - \sum_{k \in Z_i'^b} S_k'^b > 0 \tag{2.32}$$

and similarly,

$$S_{v_i}'''^a = S_{v_i}''^a - \sum_{k \in Z_{v_i}^a \backslash Z_{v_j}'^a} S_k'^a - \sum_{k \in Z_{v_j}'^a} S_k'^a \tag{2.33}$$

We want to show that $S_{v_i}'''^{a} > 0$.

Note that
$$\sum_{k \in Z^{a}_{v_i} \backslash Z'^{a}_{v_j}} S^{k'a} = \sum_{k \in Z_j{}^{b} \backslash Z'_i{}^{b}} S'_k{}^{b} \tag{2.34}$$

since these nodes are not in the unique cycle and hence $S'_k{}^{a} = S'_k{}^{b}$.

From Equation (2.24), we also have

$$S''_{v_i}{}^{a} = -S_i{}^{b} + S_p{}^{b} . \tag{2.35}$$

Subtracting Equation (2.32) from Equation (2.33) and using Equations (2.34) and (2.35), we get

$$\Delta = S'''_{v_i}{}^{a} - S'''_{v_j}{}^{b} = -S_i{}^{b} + S_p{}^{b} - \sum_{k \in Z'^{a}_{v_j}} S'_k{}^{a} - S''_j{}^{b} + \sum_{k \in Z'_i{}^{b}} S'_k{}^{b} \tag{2.36}$$

If $\Delta \geq 0$, then $S'''_{v_i}{}^{a} > 0$ by the induction hypothesis. On the other hand, if $\Delta < 0$, then in order to show that $S'''_{v_i}{}^{a} > 0$, it must be true that

$$S'''_{v_i}{}^{a} = S'''_j{}^{b} + \Delta > 0, \tag{2.37}$$

that is,

$$S''_j{}^{b} - \sum_{k \in Z_j{}^{b} \backslash Z'_i{}^{b}} S'_k{}^{b} - \sum_{k \in Z'_i{}^{b}} S'_k{}^{b} - S_i{}^{b} + S_p{}^{b} - \sum_{k \in Z'^{a}_{v_j}} S'_k{}^{a} - S''_j{}^{b} + \sum_{k \in Z'_i{}^{b}} S'_k{}^{b} > 0 \tag{2.38}$$

Simplifying terms, we have $S'''_{v_i}{}^{a} > 0$ if

$$S_p{}^{b} - S_i{}^{b} - \sum_{k \in Z_j{}^{b} \backslash Z'_i{}^{b}} S'_k{}^{b} > \sum_{k \in Z'^{a}_{v_j}} S'_k{}^{a}. \tag{2.39}$$

On the contrary, assume that

$$S_p{}^{b} - S_i{}^{b} - \sum_{k \in Z_j{}^{b} \backslash Z'_i{}^{b}} S'_k{}^{b} \leq \sum_{k \in Z'^{a}_{v_j}} S'_k{}^{a}. \tag{2.40}$$

From our selection of node j, we know that $S'^{a}_{v_j} > 0$, thus from Equations (2.17) and (2.23) we have

$$\sum_{k \in Z'^{a}_{v_j}} S'^{a}_{k} = S^{a}_{v_j} = - S''^{b}_{j} + S_p{}^{b}. \tag{2.41}$$

Hence

$$S''^{b}_{j} - S_i{}^{b} - \sum_{k \in Z_j{}^{b} \setminus Z'^{b}_{i}} S'^{b}_{k} \le 0. \tag{2.42}$$

Since $S_i{}^{b} \le \sum_{k \in Z'^{b}_{i}} S'^{b}_{k}$, Equation (2.42) is a contradiction to Equation (2.32). Moreover, using Equations (2.32) and (2.41) we have

$$S^{b}_{p} - \sum_{k \in Z'^{b}_{i}} S'^{b}_{k} - \sum_{k \in Z_j{}^{b} \setminus Z'^{b}_{i}} S'^{b}_{k} > \sum_{k \in Z'^{a}_{v_j}} S'^{a}_{k}. \tag{2.43}$$

Thus, we have shown that $S'''^{a}_{v_i} > 0$ if there exists a node $j \in V_B{}^{b}$ such that $i,j, \in P_f{}^{b} \setminus P_p{}^{b}$ with $D(j)<D(i)$, and $S'^{a}_{v_j} > 0$. If $S'^{a}_{v_j} \le 0$, for all $j \in P_i{}^{b} \setminus P_p{}^{b} \cap V_B{}^{b}$, then we can choose the node j such that $D(j) = \min \{D(k) \; ; \; k \in P_i{}^{b} \setminus P_p{}^{b} \cap V_B{}^{b}\}$. Thus we have

$$\sum_{k \in Z'^{a}_{v_j}} S'^{a}_{k} = \sum_{k \in Z'^{a}_{p}} S'^{a}_{k} = \sum_{k \in Z_p{}'^{b} \setminus Z_j{}^{b}} S'^{b}_{k}. \tag{2.44}$$

Substituting Equation (2.44) into (2.43), we have

$$S^{b}_{p} - \sum_{k \in Z'^{b}_{i}} S'^{b}_{k} - \sum_{k \in Z_j{}^{b} \setminus Z'^{b}_{i}} S'^{b}_{k} - \sum_{k \in Z'^{b}_{p} \setminus Z_j{}^{b}} S'^{b}_{k} > 0 \tag{2.45}$$

which agrees with Equation (2.17) and the fact that $S'^{b}_{p} > 0$. This completes the proof for case b.

c) In this case, we will consider the node $f \in V_B{}^a$. From Equation (2.25), and Lemma 2.4,

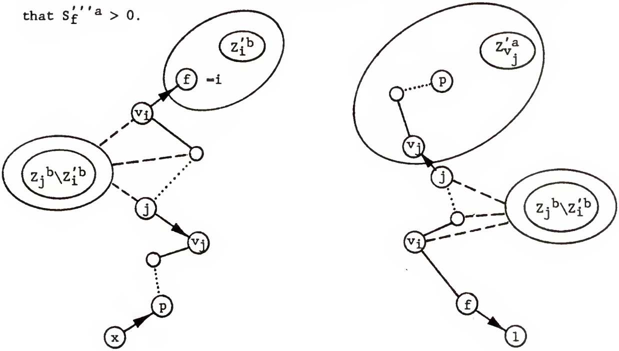
$$S_f^{\prime\prime a} = S_p^b > 0. \tag{2.46}$$

To show $S_f^{\prime\prime\prime a} > 0$, we will consider two subcases :

First, we will consider the subcase when $f \in V_F{}^b$. (Figure 2.8) In this case, we can choose $i=f$ as we did in case b). Then from Equation (2.43) we have,

$$S_p^b - \sum_{k \in Z_i^{\prime b}} S_k^{\prime b} - \sum_{k \in Z_j{}^b \backslash Z_i^{\prime b}} S_k^{\prime b} - \sum_{k \in Z_{v_j}^{\prime a}} S_k^{\prime a} > 0. \tag{2.47}$$

Since $Z_f{}^a = Z_{v_j}^{\prime a} \cup Z_i^{\prime b} \cup Z_j{}^b \backslash Z_i^{\prime b}$, and $S_k^{\prime b} = S_k^{\prime a}$ for those nodes that are not on the unique cycle, and $S_f^{\prime\prime a}$ equals S_p^b, Equation (2.47) implies that $S_f^{\prime\prime\prime a} > 0$.



Before the pivot After the pivot

Figure 2.8. The nodes between f and p before and after the Pivot

The second subcase is when $f \in V_B{}^b$ (Figure 2.9). We select node i such that $D(i) = \max\ \{D(i) : i \in (P_f{}^b \backslash P_p{}^b) \cap V_F{}^b\}$.

If for each node $k \in (P_f{}^b \backslash P_i{}^b) \cap V_B{}^b$, $S'^a_{v_k} \leq 0$, where $v_k = PR(k)$ in T^b, then choosing any $j \in (P_i{}^b \backslash P_p{}^b) \cap V_B{}^b$ and using Equation (2.45), we have $S'''^a_f > 0$.

However, if for a node $k \in (P_f{}^b \backslash P_i{}^b) \cap V_B{}^b$, $S'^a_{v_k} > 0$, then from Equations (2.17) and (2.23), we have

$$\sum_{i \in Z'^a_{v_k}} S'^a_i = S^a_{v_k} = -S''^b_k + S^b_p \tag{2.48}$$

Since $S'''^b_k > 0$, we have

$$S''^b_k - \sum_{i \in Z_k{}^b \backslash Z_f{}^b} S'^b_i - \sum_{i \in Z_f{}^b} S'^b_i > 0 \tag{2.49}$$

Substituting Equation (2.48) into (2.49) we have

$$S^b_p - \sum_{i \in Z'^a_{v_k}} S'^a_i - \sum_{i \in Z_k{}^b \backslash Z_f{}^b} S'^b_i - \sum_{i \in Z_f{}^b} S'^b_i > 0. \tag{2.50}$$

Since S''^a_f equals S^b_p, and $Z_f{}^a = Z_k{}^b \cup Z'^a_{v_k}$, the proof follows.

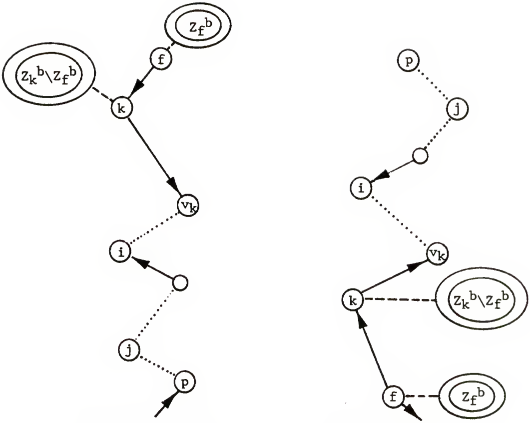


Figure 2.9. Nodes between f and p before and after the Pivot

d) Consider the nodes $i \in P_1{}^b \backslash P_\theta{}^b \cap V_B{}^b$ (Figure 2.10). The condition i) remains true since from Equation (2.22), we have

$$S_i''^a = S_i''^b + S_p^b \ . \tag{2.51}$$

Since $S_i''^b > 0$, from the assumption and $S_p^b > 0$ from Lemma 2.4, we have $S_i''^a > 0$.

To show that condition ii) remains true, we let k be the node such that

$D(k) = \max \{ D(t),\ t \in P_1{}^b \backslash P_\theta{}^b \cap V_F,\ S_t'^b > 0, \text{ and } S_t'^a > 0 \}$. For all those i on the path $P_k \backslash P_\theta$, $T_i'''^b$ and $T_i'''^a$ are the same, therefore $S_i'''^a = S_i'''^b$.

Now, we let j be the node such that $j \in P_f{}^a \backslash P_k{}^a \cap V_F$, $s_j'^b \leq 0$ and $s_j'^a > 0$. For all nodes w on $P_f{}^a \backslash P_j{}^a$, $Z_w{}^a = Z_w{}^b \cup Z_f{}^a$. Since $s_w''^a - s_w''^a$ equals $s_p{}^b$, it follows that

$$s_w'''^a - s_w'''^b = s_p{}^b - \sum_{k \in Z_f{}^a} s_k'^a = s_f'''^a > 0. \tag{2.52}$$

For nodes i on $P_j{}^a \backslash P_k{}^a$, $s_i'''^a - s_i'''^b = -s_j'^b > 0$.

This completes the proof of the lemma. ■

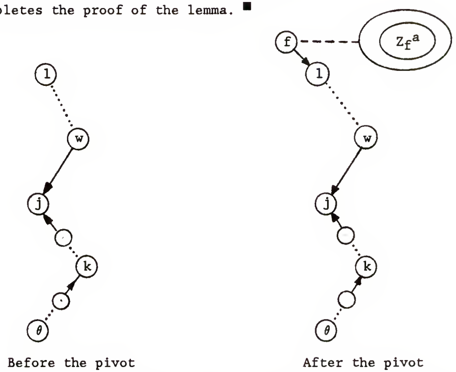


Figure 2.10. Nodes between 1 and θ before and after the pivot

What we have shown with Lemma 2.5, is that our selection rule of the leaving arc maintains a strongly feasible tree, a feasible tree in which every arc with a flow of zero is a forward arc, introduced by Cunningham [19] and Barr, Glover and Klingman [20]. Moreover, in the algorithm we always have positive flow on all the backward arcs.

Theorem 2.1: When the algorithm terminates, the final tree T^* obtained by the algorithm possesses a feasible (and optimal) solution to (MCP).

Proof: Since for each node $i \in V_B$, we have $S_i'' > 0$ from lemma 2.5, each backward arc in T^* will have a positive flow. Each forward arc that defines an induced subtree for a node will have a non-negative flow from the termination criterion that $S_i \leq 0$, $i \in V_F$. Thus, with Lemma 2.1, the proof follows. ■

2.4.8 Convergence and the Computational Bound of the Algorithm

Definition 2.2: Given an initial dual feasible solution and its corresponding tree T^0, the initial total deficiency S_{max} is defined as follows:

$$S_{max} = \sum_{i \in Q} S_i' \tag{2.53}$$

During the course of the algorithm S_{max} is defined in the same manner.

We will also present another way of determining the S_{max} value. We will use this definition in the later sections. Noting that

$$S_i = S_i' + \sum_{j \in Z_i} S_j' \tag{2.54}$$

if $S_i' > 0$, then the deficiency in a given subtree T_i can be expressed as S_i.

Define a set $K \subseteq Q$ where $K = \{ j : S_j' > 0$, and there exists no other node $k \in P_j$ with $S_k' > 0\}$. That is the set K contains those nodes in Q which are closest to the root in every path. For example, in the

problem in Figure 2.11, if Q= { 2,4,6,7,9}, then K= {2,6}. Then, we can define the S_{max} value as

$$S_{max} = \sum_{i \in K} S_i. \tag{2.55}$$

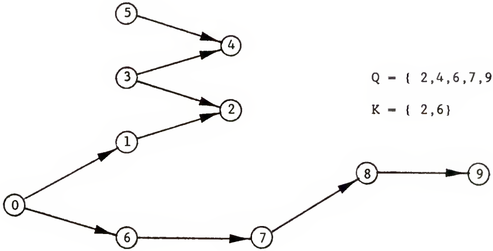


Figure 2.11. An Example for the Definition of Set K

At iteration k+1, let a pivot involving T_p bring an arc $(f,l) \notin E_T$ into the tree. Consider the set $N_1 = \{i: i \in P_1 \cap V_F\}$.

<u>Lemma 2.6</u>. If all the nodes $\alpha \in N_1$ has $S_\alpha'^b < 0$, then the S_{max} value will reduce by at least one after the pivot. On the other hand, if there exists a node $\alpha \in N_1$ with $S_\alpha'^b \geq 0$, then S_{max} value will remain the same after the pivot.

<u>Proof</u>: Before the pivot, the total deficiency in the subtree T_p is,

$$S_p^b = \sum_{i \in Z_p'^b} S_i'^b = \sum_{i \in V_{pS}^b} a_i - \sum_{j \in V_{pD}^b} b_j \tag{2.56}$$

since $S_p' > 0$.

Let $z \in N_1$ such that $D(z) = \max_{\alpha \in N_1} \{D(\alpha)\}$.

In proving Lemma 2.6, we will consider four cases.

<u>Case a)</u> $S_z'^b < 0$, but $S_z'^a > 0$.

The total deficiency in the separate induced subtrees T_p^b and T_z^b before the pivot equals

$$\sum_{i \in Z_p'^b} S_i'^b + \sum_{i \in Z_z^b} S_i'^b \tag{2.57}$$

The total deficiency in the subtree $T_z^a = T_z^b \cup T_p^b \cup \{(f,1)\}$ after the pivot equals

$$\begin{aligned} S_z^a &= S_p^b + S_z^b \\ &= \sum_{i \in Z_p'^b} S_i'^b + \sum_{i \in Z_z^b} S_i'^b + S_z'^b \end{aligned} \tag{2.58}$$

Since $S_z'^b < 0$, the total deficiency in the combined subtree T_z^a decreases by an amount $|S_z'^b|$. However, since $S_z'^b < 0$, the subtree $T_z'^b$ had not been pruned from other subtrees, $T_\alpha'^b$ $\alpha \in N_1$ by the procedure PRUNE1. But, $S_z'^a > 0$ indicates $T_z'^a$ will be pruned from other subtrees $T_\alpha'^a$, $\alpha \in N_1$. This pruning will increase some $S_\alpha'^a$ values by an amount of $|S_z'^b|$. If there exists a node $\alpha_k \in N_1$ with $S_{\alpha_k}'^b \geq 0$, such that $D(\alpha_k) = \max \{ D(\alpha): \alpha \in N_1, S_\alpha'^b \geq 0\}$, then the contribution of node α_k to the S_{max} value will increase by the amount $|S_z'^b|$. Since $T_{\alpha_k}'^b$ was pruned from other induced subtrees T_α' before the pivot, and it will be pruned after the pivot, other S_α' values will not change, since $Z_\alpha^a = Z_\alpha^b$. Hence

the total S_{max} value remains the same. On the other hand, consider the case that for all the nodes $\alpha \in N_1$, $S_\alpha'^b < 0$. Let $\alpha_1, \alpha_2, \ldots \alpha_n$ be the subset of nodes in N_1 with decreasing depth functions such that $S_{\alpha_i}'^a > 0$, $i=1,\ldots,n$ after the pruning procedure.

We need to show that $\sum_i S_\alpha'^a < | S_{z_i}'^b |$.

$$\text{Note that } S_{\alpha_n}^a = S_{\alpha_n}'^a + S_{\alpha_{n-1}}'^a + \ldots + S_{\alpha_1}'^a + S_z^a + \sum_{i \in Z_{\alpha_n}^a \backslash Z_z'^a} S_i'^a \tag{2.59}$$

$$S_{\alpha_n}^b = S_{\alpha_n}'^b + \sum_{i \in Z_{\alpha_n}^b \backslash Z_z^b} S_i'^b + \sum_{i \in Z_z^b} S_i'^b \tag{2.60}$$

and

$$S_{\alpha_n}^a = S_{\alpha_n}^b + S_p^b \qquad \text{(from Equation (2.21))} \tag{2.61}$$

Since the S_i' values for the nodes that are not in the unique cycle do not change after the pivot, we have,

$$\sum_{i \in Z_{\alpha_n}^b \backslash Z_z^b} S_i'^b = \sum_{i \in Z_{\alpha_n}^a \backslash Z_z'^a} S_i'^a \tag{2.62}$$

Subtracting (2.59) from (2.60), and using (2.61) and (2.62), we get

$$\sum_i S_{\alpha_i}'^a + S_z'^b = S_{\alpha_n}'^b < 0 \tag{2.63}$$

indicating that the total contribution of nodes $\alpha_1, \ldots \alpha_n$ to the S_{max} value is strictly less than $|S_z'^b|$. Thus, we have shown that for case a) if all the nodes $\alpha \in N_1$ have $S_\alpha'^b < 0$, the S_{max} value reduces by at least one.

<u>Case b)</u> $S_z'^b \geq 0$, $S_z'^a > 0$.

Since $S_z'^b \geq 0$, $z \in Z_z'^b$ and $z \in Z_z'^a$. Thus from Equations (2.57) and (2.58), we note that the total deficiency in the subtree $T_z{}^a$ remains the same. Moreover, $Z_\alpha{}^b = Z_\alpha{}^a$ implies that $S_\alpha'^a = S_\alpha'^b$ for $\alpha \neq z \in N_1$. Thus the S_{max} value remains the same.

<u>Case c)</u> $S_z'^b < 0$, $S_z'^a \leq 0$.

Total deficiency in the subtrees $T_p{}^b$ and $T_z{}^b$ equals

$$\sum_{i \in Z_z{}^b} S_i'^b + S_p^b. \tag{2.64}$$

Total deficiency in the subtree $T_z{}^a$ equals

$$\sum_{i \in Z_z{}^a \backslash Z_f{}^a} S_i'^a + \sum_{i \in Z_f{}^a} S_i'^a = \sum_{i \in Z_z{}^b} S_i'^b + \sum_{i \in Z_f{}^a} S_i'^a \tag{2.65}$$

The equality in (2.65) follows from the fact that the nodes in the subtree $T_z{}^a \backslash T_f{}^a$ are not included in the unique cycle.

From Equations (2.25) and (2.18), we have

$$S_p{}^b = S_f''^a = \sum_{i \in Z_f{}^a} S_i'^a + S_f'''^a. \tag{2.66}$$

From Lemma 2.5, we have $S_f'''^a > 0$. Substituting (2.66) into (2.64) and subtracting Equation (2.65) from (2.64), we see that the deficiency in the subtrees involved in the pivot decreases by the amount $S_f'''^a$. However $S_f'''^a$ is added to $S_\alpha'^b$ value for some $\alpha \in N_1$. Using the same argument as in case a) if all the nodes $\alpha \in N_1$ have $S_\alpha'^b < 0$, then the

S_{max} value reduces by at least one. If there exists a node α with $S_{\alpha}'^{b} \geq 0$, then the S_{max} value remains the same.

<u>Case d)</u> $S_z'^{b} > 0$, $S_z'^{a} \leq 0$.

This case cannot occur because of Equations (2.17), and (2.21). Since $S_z{}^a = S_z{}^b + S_p{}^b$, $S_z{}^b$ value increases and the $S_z{}'^b$ value should increase as well.

<u>Lemma 2.7</u>. The total number of pivots without decreasing the S_{max} value is bounded by $|V|$.

<u>Proof</u>: If the S_{max} value remains the same at iteration k+1, then $\alpha \in Q^{k+1}$ and $D^{k+1}(\alpha) \leq D^{k+1}(f)$. Assume that $\alpha \in Q^{k+1}$ is the node in N_1 with the lowest depth function. Then, the next pivot will be done on $T_{\alpha}{}^{k+1}$ where $V_p{}^k \subset V_{\alpha}{}^{k+1}$ since $T_{\alpha}{}^{k+1}$ contains at least one more node than $T_p{}^k$. Thus the proof follows.

<u>Theorem 2.2</u>. The total number of dual simplex pivots of the algorithm is bounded by $M|V|$ where $M = \sum_{j \in V_S} a_j$.

<u>Proof</u>: Since $S_{max} < M$ initially, and there can be at most $|V|$ pivots before S_{max} is reduced by at least one, the proof follows.

Now, we will state the complexity of the algorithm DUAL in terms of arithmetic operation. We note that each pivot requires at most $O(|V|^2)$ operations since at most $|V|^2$ comparisons are needed to find the value of Δ, given by Equation (2.11), at each pivot. However with

the following implementation we can reduce the complexity in terms of operations.

We let k_0 be the iteration in which S_{max} value has been decreased, and k_{n+1} be the next iteration in which S_{max} value is decreased again. Thus, $k_1,\ldots,k_n$ are the iterations in which S_{max} remains the same. Then $T_{p1},\ldots,T_{pn}$ are the subtrees involved in iterations k_1 to k_n. We note that $T_{p1} \subset T_{p2} \subset \ldots \subset T_{pn}$.

At iteration k_1, for each node $i \in V_{p1}$, we find $\Delta_i = \min \{\Delta_{ij} : j \in V \backslash V_{p1}\}$ and store the value of Δ_i and the node $j \in V \backslash V_{p1}$ which yielded the minimum. Then Δ is the minimum of the Δ_i values for $i \in V_{p1}$. Since we update the dual variables, w_i for $i \in V_{p1}$, by adding Δ, node j that yielded the minimum Δ_i value for each node i does not change for iteration k_2, and Δ_i values are decreased by Δ for every $i \in V_{p1}$. Thus, starting from iteration k_1, if we store the Δ_i value and the name of the node j which yields the Δ_i value, the only extra comparisons are those to find the Δ value which is the minimum of the Δ_i values. Note that we have to replace a name and a value for some node i. However, each node i needs to be "scanned" only once until the S_{max} value reduces at iteration k_{n+1}. Thus, at most $|V|^2$ comparisons are needed for iterations k_1 to k_n.

Theorem 2.3. The algorithm DUAL requires at most $M|V|^2$ arithmetic operations, where $M = \sum_{j \in V_S} a_j$.

2.5 Scaling Method for Minimum Cost Flow Problems

In this section, we scale the supplies and demands of the minimum cost flow problem similar to that of Edmonds and Karp's approach [28].

We define τ by $2^{\tau-1} \leq \max_{\substack{i\in V_S \\ j\in V_D}} \{ a_i, b_j \} \leq 2^{\tau}$. (2.67)

In this scaling method, we will solve τ+1 minimum cost flow problems each with different updated supply and demand values. We will refer to each different problem as a subproblem. Output of one subproblem is input to the next subproblem as a starting basic feasible solution.

In the 0^{th} subproblem, the supplies and demands are scaled as

$$a_i^0 = \lceil a_i/2^{\tau} \rceil = 1 \quad \text{and} \quad b_j^0 = \lceil b_j/2^{\tau} \rceil = 1 \quad \text{for all } i\in V_S,\ j\in V_D \tag{2.68}$$

where $\lceil x \rceil$ is the smallest integer greater than or equal to x. For the z^{th} subproblem, the supplies and demands are updated as

$$a_i^z = \lceil a_i/2^{\tau-z} \rceil \quad \text{and} \quad b_j^z = \lceil b_j/2^{\tau-z} \rceil \quad \text{for all } i\in V_S,\ j\in V_D. \tag{2.69}$$

If we use the leaving-arc selection rule of algorithm DUAL in solving the problem in a given subproblem , it may be possible to have some nodes $i\in V_B$ with $S_i''' < 0$ in the following subproblems because of the changes in a_i and b_j values. To circumvent this, the leaving arc selection will be modified so that Lemma 2.5 holds true in all the subproblems. That is, each backward arc in the rooted tree T will have a positive flow throughout the algorithm. Since the algorithm selects only forward arcs as the leaving arc, with the modified leaving arc

selection, the convergence of the algorithm is guaranteed. The modified rule is as follows:

Assume that at subproblem z, T_p is the subtree selected to be pivoted on by the algorithm DUAL. Before determining the entering arc, the scaling algorithm first checks $S_p'^k$, $k > z$, in the remaining subproblems with a_i^k and b_j^k as supplies and demands. If in any remaining subproblem t, $S_p'^t \leq 0$, then this subtree is not selected for pivoting. (Note here that if we pivot on T_p and the subtree remains unchanged until subproblem t, then we will have a backward arc with nonpositive flow in subproblem t.) Here, by remaining subproblems, we mean all future minimum cost flow problems with

$$a_i^k = \lceil a_i/2^{\tau-k} \rceil \quad \text{and} \quad b_j^k = \lceil b_j/2^{\tau-k} \rceil \text{ for } k > z. \tag{2.70}$$

Let Q^z be the set of nodes $j \in V_F$ with $S_j'^z, S_j'^{z+1}, \ldots, S_j'^\tau > 0$. Also let $K^z \subseteq Q^z$ be the set of nodes k such that there exists no other node $j \in P_k$ and $j \in Q^z$.

Lemma 2.7. Initial S_{max} value at the beginning of subproblem 0 is less than $|V|$.

Proof: In the 0^{th} subproblem, we have $a_i^0 = b_j^0 = 1$ for $i \in V_S$, $j \in V_D$, a possibly unbalanced minimum cost flow problem, and the initial S_{max} value is bounded by $|V|$.

Note here that the unbalanced problem does not create any difficulties in this algorithm since a subtree T_i with $S_i' > 0$ may not be pivoted on due to the altered leaving arc selection rule. Moreover, we may never have a subtree T_i, with $S_i' > 0$, and T_i includes all the demand nodes and S_i' remains positive in all the remaining subproblems

since at least in the $(\tau)^{th}$ subproblem we have a balanced minimum cost flow problem.

Throughout Sections 2.5 and 2.6, we will use superscripts A and B in the following manner. $(.^z)^A$ indicates the subtree (.) or a value (.) at the end of subproblem z, and $(.^z)^B$ indicates the subtree (.) or a value (.) at the beginning of subproblem z.

For any subproblem $z \geq 1$, we have

$$2a_i^{z-1} - 1 \leq a_i^z \leq 2a_i^{z-1} \qquad i \in V_S \tag{2.71}$$

$$2b_j^{z-1} - 1 \leq b_j^z \leq 2b_j^{z-1} \qquad j \in V_D. \tag{2.72}$$

Note that $(S_\alpha^z)^B$, the S_α value $\alpha \in V_F$ at the beginning of subproblem z, equals twice $(S_\alpha^{(z-1)})^A$, the S_α value at the end of subproblem (z-1), if all b_j^z and a_i^z, $i \in T_{\alpha S}$, $j \in T_{\alpha D}$ assume their largest possible values in (2.71) and (2.72). In general, we can write

$$(S_\alpha^z)^B = 2(S_\alpha^{(z-1)})^A + |(N_D)_\alpha^z| - |(N_S)_\alpha^z| \tag{2.73}$$

where $(N_D)_\alpha^z = \{\ j : j \in V_{\alpha D}$ and $b_j^z = 2b_j^{z-1} - 1\ \}$ and $(N_S)_\alpha^z = \{\ i : i \in V_{\alpha S}$ and $a_i^z = 2a_i^{z-1} - 1\ \}$.

With the following lemma, we will show that the S_{max}^z value, the total deficiency at the beginning of subproblem z is bounded by a polynomial of $|V|$. However, before stating the lemma, we will consider the case when for a node $\alpha \in V_F$, $(S_\alpha'^{z-1})^A > 0$. Since $(S_\alpha'^{z-1})^A > 0$, it implies that there exists a subproblem $t > z-1$ where $(S_\alpha'^{t})^B \leq 0$. Otherwise, node α would be in Q^{z-1} and subproblem (z-1) would not be terminated. Therefore, even though $(S_\alpha'^{z})^B$ may be positive at the beginning of subproblem z, we will not pivot on that subtree, since $\alpha \notin Q^z$, thus $(S_\alpha'^{z})^B$ will not be added to the S_{max}^z value for that subproblem. In such a case, in order to eliminate positive $(S_\alpha'^{z})^B$ in

calculating the S^z_{max} value, we can temporarily add a dummy demand node, d, to T'^z_α with $b_d = (S'^z_\alpha)^B$, thus making T'^z_α not eligible for pivot at the beginning of subproblem z. After adding the dummy demand node, we can recalculate the $(S_k{}^z)^B$ values for those nodes $k \in V_F$ on the path P_α. At the end of subproblem z, we can remove the dummy node d. Keeping this discussion in mind, we can define S^z_{max} value, the total deficiency at the beginning of subproblem z as follows:

$$S^z_{max} = \sum_{i \in Q^z} (S'^z_i)^B \tag{2.74}$$

or

$$S^z_{max} = \sum_{i \in K^z} [(S_i{}^z)^B - \Sigma\, [(S'^z_j) : j \in T_i,\ S'^z_j > 0 \text{ but } j \notin Q^z]] \tag{2.75}$$

<u>Lemma 2.8</u>. S^z_{max}, total deficiency at the beginning of subproblem z, $z \geq 1$, is bounded by $|V_D|$.

<u>Proof</u>: In this proof, we will consider three cases.

If $\alpha \in K^z$, and $(S^{z-1}_\alpha)^A \leq 0$, then from Equation (2.73), $(S^z_\alpha)^B$ increases the most if $(N_D)^z_\alpha = V_{\alpha D}$, and $(N_S)^z_\alpha = \phi$. Thus $(S^z_\alpha)^B$ is bounded by $|V_{\alpha D}|$.

If $\alpha \in K^z$, and $(S^{z-1}_\alpha)^A > 0$, but $(S'^{z-1}_\alpha)^A \leq 0$, from Equation (2.17) note that

$$(S^{z-1}_\alpha)^A = (S'^{z-1}_\alpha)^A + \sum_{i \in (Z^{z-1}_\alpha)^A} (S'^{z-1}_i)^A. \tag{2.76}$$

Substituting Equation (2.76) into (2.73), we have

$$(S^z_\alpha)^B = 2(S'^{z-1}_\alpha)^A + 2 \sum_{i \in (Z^{z-1}_\alpha)^A} (S'^{z-1}_i)^A + |(N_D)^z_\alpha| - |(N_S)^z_\alpha| \tag{2.77}$$

Even though the value $2 \sum_{i\in(Z_\alpha^{z-1})^A} (S_i'^{z-1})^A$ may be positive, it will not be added to the S_{max}^z value. Thus $(S_\alpha^z)^B$ value that will contribute to the S_{max}^z value increases the most if $(N_D)_\alpha^z = V_{\alpha D}$, and $(N_S)_\alpha^z = \phi$. Thus $(S_\alpha^z)^B$ is bounded by $|V_{\alpha D}|$.

If for a node α, $(S_\alpha^{z-1})^A > 0$, and $(S_\alpha'^{z-1})^A > 0$, it implies that there exists a subproblem t such that $(S_\alpha'^{t})^B \le 0$, thus $\alpha \notin Q^z$ or K^z.

Since for $i,j\in K^z$, $V_i \cap V_j = \phi$, and S_{max}^z is the sum of $(S_\alpha^z)^B$ values for $\alpha\in K^z$ it follows that $S_{max}^z \le |V_D|$. ■

<u>Corollary 2.1</u>. For $i\in Q^z$, $(S_i'^{z})^B$ is bounded by $|V_{iD}'^{z}|$.

<u>Proof</u>: From Lemma 2.8, we know that for a node $\alpha\in K^z$, $(S_\alpha^z)^B$ is bounded by $|V_{\alpha D}|$. The subtree $(T_\alpha^z)^B$ is the union of the subtrees $(T_i'^{z})^B$, $i\in Q^z \cap T_\alpha^z$. Since $i\in Q^z$, we know that $(S_i'^{z-1})^A \le 0$. Thus, those demand nodes in $(N_D)_\alpha^z \cap (T_i'^{z})^B$ that increase $(S_\alpha^z)^B$ value will also increase the $(S_i'^{z})^B$ value as well. ■

<u>Theorem 2.4</u>. Total number of pivots in the scaling algorithm is bounded by $(\tau+1)|V|^2$. Moreover, the algorithm requires $O[(\tau+1)|V|^3]$ arithmetic operations.

<u>Proof</u>: Since we solve $(\tau+1)$ subproblems each with initial deficiency of at most $|V|$ by using Theorems 2.2 and 2.3, the proof follows. ■

2.6 A Different Scaling Algorithm for Minimum Cost Flow Problems

In the scaling algorithm presented in Section 2.5, for each pivot at subproblem z we have to calculate the S_i', $i \in V_F$ values for all the remaining subproblems $z+1, \ldots, \tau$, and this increases the computational time for each pivot.

In this section, we present another scaling algorithm in which the leaving-arc selection rule is changed so that we would require less computational effort at each pivot. In the new arc selection rule only the S_i' values, $i \in V_F$, for the current subproblem z and the τth subproblem are considered. Since we do not consider the intermediate subproblems $(z+1, \ldots, \tau-1)$, we may have some backward arcs with a nonpositive flow in an intermediate subproblem. That is, if we pivot on a subtree T_p in subproblem z while $S_p'^{\tau} > 0$, but $S_p'^{t} \leq 0$ for subproblem t, $z < t < \tau$, and if T_p remains unchanged until subproblem t, we may have $S_p'''^{t} \leq 0$ in subproblem t, which is a contradiction to Lemma 2.5 which states that all $S_i''' > 0$ throughout the algorithm. Thus, for any subproblem $z \neq \tau$, we can have subtrees T_i', $i \in V_F$, that are not strongly feasible. The only subproblem in which the subtrees remain strongly feasible is the τth subproblem.

In a strongly feasible tree, the deficiency caused by a node $i \in V_F$ was simply its S_i' value when $S_i'>0$. However, when there exists some backward arcs with nonpositive flows in the tree, then the deficiency caused by a node i has to be redefined. In order to define this deficiency at subproblem z, we will define a subtree $\overline{T}_i{}^z$ and its associated $\overline{S}_i{}^z$ value for $i \in V_F$.

Let $Q_1{}^z$ be the set of nodes with $S_i'^z > 0$, $i \in V_F$.

Given a subtree $T_i'^z = (V_i'^z, E_i'^z)$ for $i \in Q_1{}^z$, we define a subtree $\bar{T}_i{}^z = (\bar{V}_i{}^z, \bar{E}_i{}^z)$ of $T_i'^z$ by the following procedure, PRUNE3. In PRUNE3, starting from the tip nodes of subtree $T_i'^z$, we eliminate the subtrees $T_j'''^z$, $j \in V_i'^z \cap V_B$ with $S_j{}'''^z \leq 0$.

PROCEDURE "PRUNE3"

INPUT : $T_i'^z = (V_i'^z, E_i'^z)$, for $i \in Q_1{}^z$, and $D(k)$, $k \in V_i'^z \cap V_B$.

OUTPUT : A subtree $\bar{T}_i{}^z = (\bar{V}_i{}^z, \bar{E}_i{}^z) \subseteq T_i'^z$ with the property that for each $j \in \bar{V}_{iB}$, $S_j'''^z > 0$, and its associated $\bar{S}_i{}^z$ value.

BEGIN

INITIALIZE : Set $\bar{T}_i{}^z \leftarrow T_i'^z$, $R = \{ j \in V_i'^z \cap V_B : S_j'''^z \leq 0 \}$.

$\bar{Z}_i = \emptyset$.

WHILE $R \neq \emptyset$ DO

Find $j \in R$ such that $D(j) \geq D(k)$, for all $k \neq j \in R$.

Obtain $T_j'''^z = (V_j'''^z, E_j'''^z)$.

Set $\bar{T}_i{}^z = (\bar{V}_i{}^z \backslash V_j'''^z, \bar{E}_i{}^z \backslash E_j'''^z \cup \{(j, PR(j))\})$.

Let $M \subseteq \bar{Z}_i{}^z$ be the set of nodes m in $\bar{Z}_i{}^z$ whose unique path to node 0 passes through node j.

Set $\bar{Z}_i{}^z \leftarrow \bar{Z}_i{}^z \cup \{j\} \setminus M$

Update $S_j'''^z$, $j \in \bar{V}_i \cap V_B$ and R.

ENDWHILE

Compute
$$\bar{S}_i{}^z = \sum_{j \in \bar{V}_{iS}} a_j - \sum_{j \in \bar{V}_{iD}} b_j \tag{2.78}$$

where $\bar{V}_{iS}$ and $\bar{V}_{iD}$ are the sets of supply and demand nodes in $\bar{T}_i$, respectively.

END

Note that the subtree $\bar{T}_i^z \subseteq T_i'^z$ is obtained when all the backward arcs with nonpositive flows are removed from $T_i'^z$. $\bar{Z}_i^z$ is the set of nodes j, $j \in V_B$ whose subtrees are pruned from subtree $T_i'^z$ in subproblem z. Thus,

$$\bar{S}_i^z = S_i'^z + \sum_{j \in \bar{Z}_i^z} |S_j'''^z| \quad \text{for } i \in Q_1^z. \tag{2.79}$$

If for a node $i \in Q_1^z$, there exists some nodes $j \in V_i' \cap V_B$ such that $S_j'''^z \leq 0$, then we define the contribution of node $i \in Q_1^z$ to the total deficiency to be

$$S^z_{max,i} = \bar{S}_i^z + |\bar{Z}_i^z|. \tag{2.80}$$

On the other hand, if for a node $i \in Q_1^z$ there exists no node $j \in V_i' \cap V_B$ such that $S_j'''^z \leq 0$, then the contribution of node i to the total deficiency is $S^z_{max,i} = \bar{S}_i^z = S_i'^z$.

One may interpret the $S^z_{max,i}$ value for $i \in Q_1^z$ as the deficiency of an analogous strongly feasible subtree obtained from subtree $T_i'^z$ by replacing each subtree $T_j'''^z$, $j \in \bar{Z}_i$, with a supply node that has a weight of 1.

When we remove all subtrees $T_i'^z$ for $i \in Q_1^z$ from the tree T^z, we have a remaining subtree that has no negative flows on forward arcs, but may have some nonpositive flows on backward arcs. We will call this remaining subtree T_R. There may be some nodes $i \in T_R \cap V_F$ such that

$$S^z_{max,i} = S_i'^z + \sum_{j \in \bar{Z}_i^z} |S_j'''^z| + |\bar{Z}_i^z|$$

can be positive. Here $\bar{Z}_i^z$ is the set of nodes in $T_i'^z$ with $S_j'''^z \leq 0$. Let Q_2^z be the set of nodes $i \in T_R \cap V_F$ satisfying this condition.

We can find the nodes in $Q_2{}^z$ in the following manner :

Set $Q_2{}^z = \emptyset$.

Let $R = \{ j \in T_R \cap V_B : S_j'''^z \leq 0 \}$.

WHILE $R \neq \emptyset$ DO

Find $j \in R$ such that $D(j) \geq D(k)$ for all $k \neq j \in R$.

Find the node $i \in P_j \cap V_F$ such that $|S_i'^z| = \min\{|S_k'^z|, i \in P_j \cap V_F\}$.

Use Procedure PRUNE3 as $T_i'^z$ as the input to find $\overline{T}_i{}^z$.

If $S_i'^z - \Sigma S_j'''^z + |\overline{Z}_i{}^z| > 0$, remove T_i' from T_R and

set $Q_2{}^z = Q_2{}^z \cup \{i\}$.

Update T_R, R, $S_i'^z$ for $i \in T_R \cap V_F$, and $S_j'''^z$ for $j \in T_R \cap V_B$.

ENDWHILE.

Again we can interpret the $S^z_{max,i}$ value for $i \in Q_2{}^z$ as the deficiency of an analogous strongly feasible subtree obtained from subtree $T_i'^z$ by replacing each subtree $T_j'''^z$, $j \in \overline{Z}_i$ with a supply node that has a weight of 1.

If we let $Q^z = Q_1{}^z \cup Q_2{}^z$, then we can define the total deficiency in subproblem z, $S'_{max}{}^z$ as

$$S_{max}{}'^z = \sum_{i \in Q^z} S^z_{max,i} \tag{2.81}$$

Example 2.4. Consider the problem in Figure 2.12 with 5 source nodes and 8 demand nodes. It is given that $a_1 = 4$, $a_2 = 5$, $a_3 = 4$, $a_4 = 11$, and $a_5 = 10$, and $b_6 = 5$, $b_7 = 4$, $b_8 = 2$, $b_9 = 3$, $b_{10} = 5$, $b_{11} = 3$, $b_{12} = 1$, and $b_{13} = 11$. Here $S_{12} = S_{12}' = 1$, $\overline{S}_{12} = 4$. $\overline{Z}_{12} = \{2,3\}$ and $S_{max,12} = 6$.

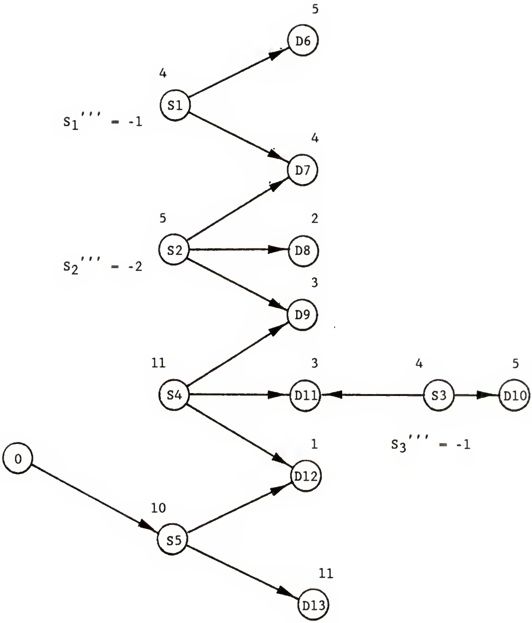


Figure 2.12. Example for the Definition of $S_{max,i}$

2.6.1. The Scaled Dual Simplex Algorithm, DUALSC

(Initialization) Calculate τ by $2^{\tau-1} \leq \max_{\substack{i \in V_S \\ j \in V_D}} \{ a_i, b_j \} \leq 2^{\tau}$.

Construct the shortest path tree of G rooted at node 0 by using costs as weights. Let w_i, $i \in V$ represent the negatives of the shortest path lengths form node 0 to node i. Let $T=(V,E_T)$ be this tree with dual variables, w_i, $i \in V$.

Set z <---- 0.

Set k <---- 0.

WHILE $z \leq \tau$ DO

Calculate

$a_i^z = \lceil a_i/2^{\tau-z} \rceil$ and $b_j^z = \lceil b_j/2^{\tau-z} \rceil$ for all $i \in V_S$, $j \in V_D$.

For each node $i \in V_F$, compute D(i), $S_i^{'\tau}$ values and $S_{max,i}^z$ values.

Let $Q^z = \{ i : i \in V_F, S_{max,i}^z > 0 \text{ and } S_i^{'\tau} > 0 \}$.

Set p' <--- 0.

WHILE $Q^z \neq \emptyset$ DO

BEGIN

k < --- k+1

(Determination of the leaving variable)

IF p'=0 THEN

Find T_p such that $D(p) = \min_{j \in Q^z} \{D(j)\}$

ELSE

Find T_p such that $D(p) = \min_{j \in Q^z \cap P_{p'}} \{D(j)\}$

where $P_{p'}$ is the set of nodes on the unique path from node p' found in iteration k-1 to node 0 in T^k.

ENDIF

Let (PR(p),p) be the arc which is dropped to obtain T_p.

(Determination of the entering variable)

Find $\Delta = \min_{\substack{i \in V_p \\ j \in V \backslash V_p \\ (i,j) \in E}} \{\Delta_{ij} = w_j - w_i + c_{ij}\} = \Delta_{fl}$

Let $w_j \leftarrow w_j + \Delta \quad j \in V_p$

$w_j \leftarrow w_j \quad j \in V \backslash V_p$

END

(Updating)

Update T, $S_{max,i}{}^z$, $S_i^{'\tau}$, Q^z.

IF $P_p \cap Q^z = \emptyset$, THEN

Set $p' = 0$

ELSE

Set $p' = p$.

ENDIF

ENDWHILE

$z \leftarrow z+1$

ENDWHILE

The current tree possesses an optimal solution to (MCP).

END

2.6.2 Computational Bound of the Algorithm

In the following lemma, we assume that we are at subproblem z, and we have done k pivots at subproblem z. To eliminate confusion between pivot indices and subproblem indices, we omit the subproblem indices. Superscripts a and b are used to indicate a value after and before the pivot, respectively.

At iteration k+1, let a pivot involving T_p bring an arc $(f,l) \notin E_T$ into the tree. Consider the set $N_1 = \{ i: i \in P_1 \cap V_F \}$ in T^b.

Lemma 2.11. If all the nodes in $\alpha \in N_1$ have $S_{max,\alpha}{}^b < 0$, then the S'_{max} value decreases after the pivot. On the other hand, if there exists a node $\alpha \in N_1$ with $S_{max,\alpha}{}^b \geq 0$, then the S'_{max} value may remain the same after the pivot.

Proof: One way of proving the lemma is to use the analogous strongly feasible tree obtained by replacing the subtrees with negative S_j''' values by supply nodes of weights one. Since we can interpret the S'_{max} value of the original tree as the S_{max} value of this analogous strongly feasible tree, we can use the same proof as in Lemma 2.6. However, we include the proof of the lemma, using the original tree.

In this proof, we will first consider the change in the $S_{max,i}$ values of nodes i in the subtree $T_p{}^b$. Afterwards, we will consider the change in the $S_{max,i}$ values of the nodes i in N_1. For all those nodes $i \in V_F$ that are not on the unique cycle, the $S_{max,i}$ values will not change.

In the subtree $T_p{}^b$, only the $S_{max,i}$ values of the nodes i that are on the path $P_f{}^b \backslash P_p{}^b$ will change by the pivot since these are the only nodes that lie on the unique cycle.

Let $v_1, \ldots, v_K$ be the nodes in $P_f^b \backslash P_p^b \cap Q^b$ such that there exists a node $j \in \bar{Z}_{vi}^b \cap (P_f^b \backslash P_p^b)$. (Figure 2.14)
These are the nodes in Q^z whose induced subtrees contain a node $j \in V_B$ on $P_f^b \backslash P_p^b$ with $S_j^{'''b} \leq 0$. We will call this node $j(v_i)$. The case when there does not exist such nodes v_i will be handled subsequently.
Let t_i be the last node of $T_j(v_i)^{'''b}$ on the path $P_f^b \backslash P_p^b$. That is, $D(t_i) = \max\{D(t) : t \in P_f^b \backslash P_p^b \cap V_j^{'''}(v_i)^b$.

We will define a subtree Γ_1^b as

$$\Gamma_1^b = \bigcup_{i \in (P_{v_1}^b \backslash P_p^b) \cap Q^b} \bar{T}_i^b.$$

In general Γ_k^b is defined as

$$\Gamma_k^b = \bigcup_{i \in (P_{v_k}^b \backslash P_{t_{k-1}}^b) \cap Q^b} \bar{T}_i^b \quad \text{for } k=2,\ldots,K.$$

Note that the subtrees Γ_k^b and Γ_{k+1}^b are separated by the subtree $T_j^{'''}(v_k)^b$. The contribution of the nodes in Γ_k^b to the $S_{max}^{'b}$ value can be expressed as

$$\sum_{i \in \Gamma_k^b \cap Q^b} \bar{S}_i^b + \sum_{i \in \Gamma_k^b \cap Q^b} |\bar{Z}_i^b|$$

$$= \sum_{i \in \Gamma_k^b \cap V_S} a_i - \sum_{j \in \Gamma_k^b \cap V_D} b_j + \sum_{i \in \Gamma_k^b \cap Q^b} |\bar{Z}_i^b|.$$

We will let $q_k = PR(j(v_k))$ for $k=1,\ldots,K$ in T^b.

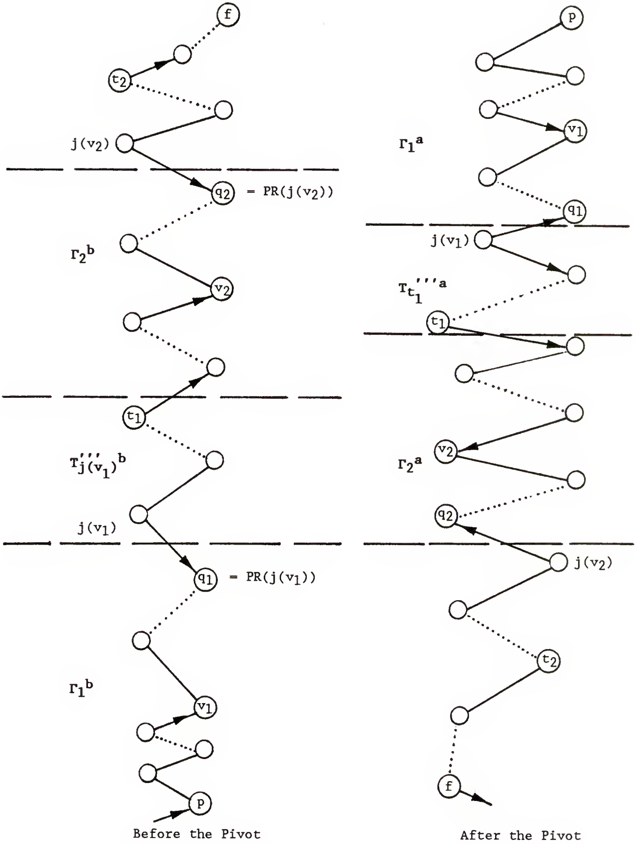


Figure 2.13. Nodes in $P_f^b \backslash P_p^b$ before and after the Pivot

After the pivot, we define $\Gamma_1{}^a$ as the subtree rooted at node q_1 such that

$$\Gamma_1{}^a = \bigcup_{i \in (P_p{}^a \setminus P_{q_1}{}^a) \cap Q^a} \overline{T}_i{}^a \quad .$$

The contribution of the nodes in $\Gamma_1{}^a$ to the $S'_{max}{}^a$ value can be expressed as

$$\sum_{i \in \Gamma_1{}^a \cap Q^a} \overline{S}_i{}^a + \sum_{i \in \Gamma_1{}^a \cap Q^a} |\overline{Z}_i{}^a|$$

$$= \sum_{i \in \Gamma_1{}^a \cap V_S} a_i - \sum_{j \in \Gamma_1{}^a \cap V_D} b_j + \sum_{i \in \Gamma_1{}^a \cap Q^a} |\ \overline{Z}_i{}^a\ |$$

Since the node sets of $\Gamma_1{}^a$ and $\Gamma_1{}^b$ are the same, we have

$$\sum_{i \in \Gamma_1{}^a \cap V_S} a_i - \sum_{j \in \Gamma_1{}^a \cap V_D} b_j = \sum_{i \in \Gamma_1{}^b \cap V_S} a_i - \sum_{j \in \Gamma_1{}^b \cap V_D} b_j$$

Moreover,

$$\sum_{i \in \Gamma_1{}^a \cap Q^a} |\ \overline{Z}_i{}^a\ | = \sum_{i \in \Gamma_1{}^b \cap Q^b} |\ \overline{Z}_i{}^b\ | - 1$$

since the node $j(v_1)$ is no longer in $\overline{Z}_i{}^a$. Thus the contribution of the nodes in $\Gamma_1{}^a$ to S'_{max} value decreases by one after the pivot.

We will now look at the nodes in $T'''_{j(v_1)}{}^b$ after the pivot.

Since the node sets of $T'''_{j(v_1)}{}^b$ and $T'''_{t_1}{}^a$ are the same, we have

$$S'''_{t_1}{}^a = S'''_{j(v_1)}{}^b < 0.$$

Thus subtree $T_{t_1}'''^{a}$ will be pruned by procedure PRUNE3 after the pivot.

In general, we can define the subtree $\Gamma_k{}^a$ k=2,...,K as the subtree rooted at node q_k as

$$\Gamma_k{}^a = \bigcup_{i \in P_{t_{k-1}}{}^a \setminus P_{q_k}{}^a \cap Q^a} \bar{T}_i{}^a .$$

The contribution of the nodes in $\Gamma_k{}^a$ $k \neq 1$ to the $S'_{max}{}^a$ value is expressed as

$$\sum_{i \in \Gamma_k{}^a \cap Q^a} \bar{S}_i{}^a + \sum_{i \in \Gamma_k{}^a \cap Q^a} |\bar{Z}_i{}^a|$$

$$= \sum_{i \in \Gamma_k{}^a \cap V_S} a_i - \sum_{j \in \Gamma_k{}^a \cap V_D} b_j + \sum_{i \in \Gamma_k{}^a \cap Q^a} | \bar{Z}_i{}^a | .$$

Since the node sets of $\Gamma_k{}^a$ and $\Gamma_k{}^b$ are the same, and

$$\sum_{i \in \Gamma_k{}^a \cap Q^a} | \bar{Z}_i{}^a | = \sum_{i \in \Gamma_k{}^b \cap Q^b} | \bar{Z}_i{}^b | ,$$

the contribution of nodes in $\Gamma_k{}^a$ to $S'_{max}{}^a$ value equals to the contribution of nodes in $\Gamma_k{}^b$ to $S'_{max}{}^b$ value.

The value $S_f'''^{a}$ can take two possible values depending on whether node f was in the subtree $\Gamma_k{}^b$ or in the subtree $T_{j(v_K)}'''^{b}$ in T^b. The change in the contribution of nodes in $T_p{}^b$ to the S'_{max} value is reflected in the $S_f'''^{a}$ value. We will consider each case separately.

Case 1. If f was in the subtree $T_{j(v_K)}^{\prime\prime\prime b}$, then

$$S_f^{\prime\prime\prime a} = S_{j(v_K)}^{\prime\prime\prime b} < 0.$$

In this case the total contribution of the nodes in T_p^b to the $S_{max}^{\prime}$ value decreases by one since for each pair of subtrees Γ_k^a and Γ_k^b, the contribution of nodes in the subtrees remains the same for k=2,...,K, and reduces by one for k=1.

Case 2. If f was in the subtree Γ_K^b, then we have

$$\sum_{i \in \Gamma_K^a \cap Q^a} \overline{S}_i^a + S_f^{\prime\prime\prime a} = \sum_{i \in \Gamma_K^a \cap V_S} a_i - \sum_{j \in \Gamma_K^a \cap V_D} b_j = \sum_{i \in \Gamma_K^b \cap Q^b} \overline{S}_i^b .$$

Thus $S_f^{\prime\prime\prime a}$ is the difference between the contribution of the nodes in Γ_K^b and Γ_K^a. Here we want to note that if we cannot find any nodes v_i, then we will have one subtree Γ_1^b and Γ_1^a with K=1.

Now we will look at the change in the contribution to the $S_{max}^{\prime}$ value of the nodes on N_1. According to the sign of $S_f^{\prime\prime\prime a}$, we will again consider two cases.

Case 1. $S_f^{\prime\prime\prime a} \leq 0$.

Note that $S_f^{\prime\prime\prime a} \leq 0$ implies that the contribution of nodes in T_p^b to $S_{max}^{\prime}$ value has already been reduced by one.

Case 1a) If there exists a node $q \in N_1$ such that $S_{max,q}^b \geq 0$, and $D(q) = \max\{D(j) : S_{max,j}^b \quad 0\}$, then $S_f^{\prime\prime\prime a}$ value will be added to $S_q^{\prime b}$ value and $S_j^{\prime\prime\prime b}$ values for $j \in T_q^{\prime b} \cap P_1$, decreasing both values by the same amount. Using Equation (2.79) we can see that $\overline{S}_q$ value remains the same. However,

$|\overline{Z}_q^a| = |\overline{Z}_q^b| + 1$, since $\overline{Z}_q^a = \overline{Z}_q^b \cup \{f\}$. Thus, $S_{max,q}^a = S_{max,q}^b + 1$. Since the contribution of nodes in T_p^b to $Smax^{\prime}$ value is reduced by one

and the contribution of node q to $Smax'$ value is increased by one, the S'_{max} value remains the same.

Case 1b) On the other hand, if all the nodes $q \in N_1$ have $S_{max,q}{}^b < 0$, then $S_{max,q}{}^a = S_{max,q}{}^b + 1 \leq 0$. Thus, node q has still no contribution to S_{max}' value. Since the contribution of nodes in $T_p{}^b$ to S_{max}' value is reduced by one, the S'_{max} value is reduced by one.

Case 2. $S_f'''^a > 0$.

Note that $S_f'''^a > 0$ implies that the contribution of nodes in $T_p{}^b$ to S'_{max} value has already been reduced by $S_f'''^a > 0$.

Case 2a) In this case, again if there exists a node $q \in N_1$ such that $S_{max,q}{}^b \geq 0$, and $D(q) = \max\{D(j) : S_{max,j}{}^b \geq 0\}$, then the $S_f'''^a$ value will added to the $S_q{}'^b$ value and $S_j'''^b$ values for $j \in T_q{}'^b \cap P_1 \cap V_B$, increasing both values by the same amount. If there exists a node $j \in T_q{}'^b \cap P_1 \cap V_B$ with $S_j'''^b \leq 0$, then we can have two subcases:

i) If $S_f'''^a \geq |S_j'''^b|$, then the $S_j'''^a$ value will be positive, and the $\overline{S}_q$ value will increase by the amount $S_f'''^a - |S_j'''^b|$. This follows from the fact that in Equation (2.79), S_q' increases by $S_f'''^a$, and $j \notin \overline{Z}_q{}^a$. Since the contribution of nodes in T_p to the S'_{max} value has reduced by the $S_f'''^a$ value, the combined contribution of the nodes in T_p and node q to the S'_{max} value will decrease by the amount of $|S_j'''^b|$.

ii) If $S_f'''^a < |S_j'''^b|$, then the $\overline{S}_q$ value remains the same, since in Equation (2.79), the S_q' and S_j''' values increase by the same amount while j remains to be in $\overline{Z}_q$. Thus, the $S_{max,q}$ value remains the same.

Since the contribution of nodes in T_p to the S'_{max} value has been reduced by the $S_f'''^a$ value, the combined contribution of the nodes in T_p and node q to the S'_{max} value will decrease by the amount of $S_f'''^a$.

If there exists no node $j \in T_q'^b \cap P_1 \cap V_B$ with $S_j'''^b \leq 0$, then the $S_{max,q}$ value increases by the amount of the $S_f'''^a$ value, thus the S_{max}' value remains the same.

Case 2b) If there does not exist a node $q \in N_1$ such that $S_{max,q}{}^b \geq 0$, however there exists some nodes $q \in N_1$ such that $S_{max,q}{}^a \geq 0$, we will consider the following:

$S_{max,q}{}^b < 0$ implies that, $S_q'^b < 0$, and $Z_q{}^b = \emptyset$ for all $q \in N_1$. This means that there are no backward arcs with nonpositive flows on the path P_1.

Let $t \in N_1$ such that $D(t) = \max \{D(q): S_{max,q}{}^a \geq 0\}$. Since $S_f'''^a$ is added to all the flows on backward arcs, $\bar{Z}_t{}^a = \emptyset$. Thus,

$$S_{max,t}{}^a = S_t'^a = S_t'^b + S_f'''^a < S_f'''^a.$$

Since the contribution of node t to the S'_{max} value increases by the value $S_t'^b + S_f'''^a$, where $S_t'^b < 0$, the combined contribution of the nodes in subtrees T_p and T_t' to the S'_{max} value decreases by the amount of $|S_t'^b|$. Using the same argument as in Lemma 2.6 case a), we can show that the increase in $S_{max,\alpha}$ values for $\alpha \in P_t$ is strictly less than $|S_t'^b|$. Thus, the S'_{max} value decreases by at least one. This completes the proof of the lemma.

Lemma 2.12. The total number of pivots without decreasing S_{max}' value is bounded by $|V|$.

Proof: Same as Lemma 2.7.

Note that in this new scaling algorithm, DUALSC, we do not pivot on the backward arcs with negative flows since in the original problem we know that there is a positive flow on that arc.

To find the computational bound of this new scaling algorithm we consider the following lemmas. In Lemma 2.13, we will give a lower bound on the $S_j'''^z$ value for a node $j \in V_B$.

Lemma 2.13: At the beginning of subproblem z, the $S_j'''^z$ values for all $j \in V_B$ is bounded from below by $-|(V_{jD}''')^z|$.

Proof: If for a node $j \in V_B$, $S_j'''^z < 0$, then we will show that $S_j'''^z > -|V_{jD}'''^z|$.

If $T_j'''^z$ is the same subtree as $T_j'''^{z+1}, \ldots, T_j'''^{\tau}$, that is $Z_j^z = Z_j^{z+1} = \ldots = Z_j^{\tau}$ even though we change the supplies and demands in each subproblem (Figure 2.14), then we can write

$$(S_j'''^{z+1}) = 2(S_j'''^z) + |(N_{jD})^{z+1}| - |(N_{jS})^{z+1}| \tag{2.82}$$

where $(N_{jD})^{z+1} = \{ k : k \in V_{jD}'''^{z+1} \text{ and } b_k^{z+1} = 2b_k^z - 1 \}$ and $(N_{jS})^{z+1} = \{ i : i \in V_{jS}'''^{z+1} \text{ and } a_i^{z+1} = 2a_i^z - 1 \}$.

In such a case, the minimum value $(S_j'''^z)$ can assume is bounded by $-|V_{jD}'''^z|$. To see this, assume on the contrary that $(S_j'''^z) \le -|V_{jD}'''^z|$. Then from Equation (2.82), the most increase in $S_j'''^k$, $k=z,\ldots,\tau$, one can achieve is when $(N_{jD})^z = V_{jD}'''^z$ and $(N_{jS})^z = \emptyset$. Thus

$$(S_j'''^{z+1}) = 2(S_j'''^z) + |V_{jD}'''^z| \le -|V_{jD}'''^z|, \tag{2.83}$$

and

$$(S_j'''^{z+2}) = 2(S_j'''^{z+1}) + |V_{jD}'''^{z}| \le -|V_{jD}'''^{z}| \ . \qquad (2.84)$$

Continuing in this manner, we get

$(S_j'''^{\tau}) \le -|V_{jD}'''^{z}|$, contradicting to the fact that $S_j'''^{\tau} > 0$, for all $j \in V_B$, that is in the original problem, we have no negative flows on the backward arcs.

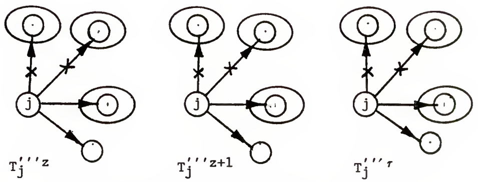


Figure 2.14 The Subtree T_j''' in the Remaining Subproblems

However, when $Z_j^{z} \neq Z_j^{z+1} \neq \ldots \neq Z_j^{\tau}$, then the subtree T_j''' changes during the remaining subproblems. Keeping this in mind, we will consider four cases shown in Figure 2.15.

Case 1 : Consider a node $i \in T_j'''$ such that $i \in Z_j^{z}$ and $i \in Z_j^{\tau}$. (For example node i in Figure 2.15a).

Case 2 : Consider a node $i \in T_j'''$ such that $i \in Z_j^{z}$ but $i \notin Z_j^{\tau}$. (For example node i in Figure 2.15b).

We will use the same proof for cases 1 and 2.

On the contrary, we will assume that, $S_j'''^z \le -|V_{jD}'''^z|$. We will consider the following subproblems, $t_1, q_1, t_2, q_2, \ldots t_K, q_K$ where $z < t_1 < q_1 < \ldots < q_K \le r$. In subproblems z to t_1-1, we have $i \in Z_j$. In subproblems t_1 to q_1-1, we have $i \notin Z_j$. In general, in subproblems t_k to q_k-1, we have $i \notin Z_j$, and in subproblems q_k to $(t_{k+1} -1)$, we have $i \in Z_j$.

Since the subtree does not change until subproblem t_1, as in the previous case we can show that at subproblem t_1-1,

$$(S_j'''^{t_1-1}) \le -|V_{jD}'''^z| \tag{2.85}$$

At subproblem t_1, subtree $T_i'^{t_1}$ is included in $T_j'''^{t_1}$. Thus we know that $S_i'^{t_1} \le 0$, and the most increase in $(S_j'''^{t_1})$ is achieved by

$$S_j'''^{t1} = 2S_j'''^{t_1-1} + |V_{jD}'''^z| + S_i'^{t_1}. \tag{2.86}$$

In the remaining subproblems until subproblem q_1, where subtree T_i' remains to be a part of subtree T_j''', nodes in $|V_{iD}'|$ help in increasing S_j''' value and S_i' value if they assume their lower bounds. Thus, the most increase in $S_j'''^{t_1+1}$ is achieved when

$$S_j'''^{t_1+1} = 2S_j'''^{t_1} + |V_{jD}'''^{t1+1}| \tag{2.87}$$

Realizing that $V_{jD}'''^{t1+1} = V_{jD}'''^z \cup V_{iD}'^{t_1+1}$, we can rewrite (2.87) as

$$S_j'''^{t_1+1} = 2S_j'''^{t_1} + |V_{jD}'''^z| + |V_{iD}'^{t_1+1}|. \tag{2.88}$$

Substituting (2.86) into (2.88) we get

$$S_j'''^{t_1+1} = 4S_j'''^{t_1-1} + 3\ |V_{jD}'''^z| + 2S_i'^{t_1} + |V_{iD}'^{t_1+1}| \tag{2.89}$$

$$= 4S_j'''^{t1-1} + 3\ |V_{jD}'''^{z}| + S_i'^{t1+1}. \tag{2.90}$$

In subproblem q_1-1, we have

$$S_j'''^{q1-1} = nS_j'''^{t-1} + (n-1)\ |V_{jD}'''^{z}| + S_i'^{q1-1} \tag{2.91}$$

where n= 2^{q1-t1}. Note here that $S_i'^{q1-1} < 0$.

Substituting (2.85) into (2.91), we get

$$S_j'''^{q1-1} \leq -|V_{jD}'''^{z}| + S_i'^{q1-1}. \tag{2.92}$$

In subproblem q_1, since $i \in Z_j^{q1}$, subtree $T_i'^{q1}$ is pruned from subtree $T_j'''^{q1}$, thus

$$S_j'''^{q1} \leq -|V_{jD}'''^{z}|. \tag{2.93}$$

Now, starting from subproblem q_1, and using the same argument as above, we can show that for subproblems q, where $i \in Z_j{}^q$, we have

$$S_j'''^{q} \leq -|V_{jD}'''^{z}|. \tag{2.94}$$

For subproblems t, where $i \notin Z_j{}^t$, we have

$$S_j'''^{t} \leq -|V_{jD}'''^{z}| + S_i'^{t} \tag{2.95}$$

where $S_i'^{t} < 0$.

Thus, for case a), where $i \in Z_j{}^\tau$, we can use (2.94) by letting τ=q, and have

$$S_j'''^{\tau} \leq -|V_{jD}'''^{z}|. \tag{2.96}$$

Similarly, for case b), where $i \notin Z_j^{\tau}$, we can use (2.95) by letting τ=t, and have

$$S_j^{'''\tau} \leq -|V_{jD}^{'''z}| + S_i^{'\tau}. \tag{2.97}$$

Both (2.96) and (2.97) is a contradiction to the fact that all $S_j^{'''}$ values are positive in the original problem. This completes the proof for cases 1 and 2.

Case 3 : Consider a node $i \in T_j^{''}$ such that $i \notin Z_j^{z}$ but $i \in Z_j^{\tau}$. (For example node 3 in Figure 2.15c).

Case 4 : Consider a node $i \in T_j^{''}$ such that $i \notin Z_j^{z}$ and $i \notin Z_j^{\tau}$. (For example node 4 in Figure 2.15d)

We will use the same proof for cases 3 and 4.

On the contrary, we will assume that, $S_j^{'''z} \leq -|V_{jD}^{'''z}|$.

At subproblem z, we can divide $V_{jD}^{'''z}$ into disjoint sets as

$$V_{jD}^{'''z} = V_i^{'z} \cup (V_{jD}^{'''z} \backslash V_i^{'z}).$$

We will consider the following subproblems, $q_1, t_1, q_2, t_2, \ldots q_K, t_K$ where $z < q_1 < t_1 < \ldots < t_K \leq \tau$. In subproblems z to q_1-1, we have $i \notin Z_j$. In subproblems q_1 to t_1-1, we have $i \in Z_j$. In general, in subproblems q_k to t_k-1, we have $i \in Z_j$, and in subproblems t_k to q_{k+1} -1, we have $i \notin Z_j$.

Since the subtree $T_j^{'''}$ does not change till subproblem q_1, we can show that at subproblem q_1-1, we have

$$S_j^{'''q_1-1} \leq -|V_{jD}^{'''z} \backslash V_i^{'z}| - |V_i^{'z}|. \tag{2.98}$$

At subproblem q_1, $i \in Z_j^{q1}$, thus $S_i'^{q1} > 0$, and subtree $T_i'^{q1}$ is pruned by procedure PRUNE2. Thus, the most increase in $S_j'''^{q1}$ is achieved when

$$S_j'''^{q1} = 2S_j'''^{q1-1} + |V_{jD}'''^{z} \backslash V_i'^{z}| + |V_i'^{z}| - S_i'^{q1}$$

$$\leq -|V_{jD}'''^{z} \backslash V_i'^{z}| - |V_i'^{z}|$$

The most increase in $S_j'''^{q1+1}$ is achieved when

$$S_j'''^{q1+1} = 2S_j'''^{q1} + |V_{jD}'''^{q1+1}|$$

$$= 2S_j'''^{q1} + |V_{jD}'''^{z} \backslash V_i'^{z}|$$

$$\leq -|V_{jD}'''^{z} \backslash V_i'^{z}| - |V_i'^{z}|$$

In subproblem t^1-1, we have

$$S_j'''^{t1-1} = S_j'''^{q1-1} + |V_{jD}'''^{z} \backslash V_i'^{z}|$$

$$\leq -|V_{jD}'''^{z} \backslash V_i'^{z}| - |V_i'^{z}|.$$

In subproblem t^1, subtree $T_i'^{t1}$ is included in $T_j'''^{t1}$. Thus we know that $S_i'^{t1} \leq 0$, and the most increase in $(S_j'''^{t1})$ is achieved by

$$S_j'''^{t1} = 2S_j'''^{t1-1} + |V_{jD}'''^{z} \backslash V_i'^{z}| + S_i'^{t1}$$

$$\leq -|V_{jD}'''^{z} \backslash V_i'^{z}| - |V_i'^{z}|.$$

Continuing in this manner, we can see that for all the remaining problems S_j''' will be less than $-|V_{jD}'''^{z} \backslash V_i'^{z}| - |V_i'^{z}|$ contradicting to the fact that all $S_j'''^{\tau}$ values are positive in the original problem. This completes the proof of the lemma.

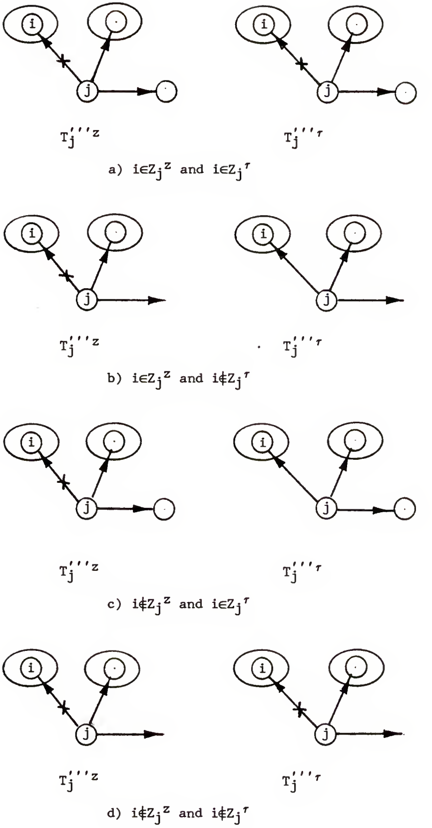


Figure 2.15 Subtrees T_j''' at Subproblems z and τ

<u>Lemma 2.14</u> : At the beginning of subproblem z, $(S_{max,\alpha}{}^{z})^{B}$ is bounded by $2|V_{\alpha D}{}'| + |V_{\alpha B}{}'|$.

<u>Proof</u> : Note that from Equations (2.79) and (2.80), we have

$$(S_{max,\alpha}{}^{z})^{B} = (S_{\alpha}{}'^{z})^{B} - \sum_{j\in\bar{Z}_{\alpha}{}^{z}} S_j'''^{z} + |\bar{Z}_{\alpha}{}^{z}|$$

$$\leq (S_{\alpha}{}'^{z})^{B} + \sum_{j\in\bar{Z}_{\alpha}{}^{z}} |V_{jD}'''^{z}| + |\bar{Z}_{\alpha}{}^{z}| \quad \text{(from Lemma 2.13)}$$

$$\leq |V_{\alpha D}'| + \sum_{j\in\bar{Z}_{\alpha}{}^{z}} |V_{jD}'''^{z}| + |\bar{Z}_{\alpha}{}^{z}| \quad \text{(from Corollary 2.1)}$$

$$\leq |V_{\alpha D}'| + \sum_{j\in\bar{Z}_{\alpha}{}^{z}} |V_{jD}'''^{z}| + |V_{\alpha B}'| \quad \text{(from the definition of the set } \bar{Z}_{\alpha}{}^{z})$$

$$\leq 2\,|V_{\alpha D}'| + |V_{\alpha B}'| \quad .\blacksquare$$

<u>Lemma 2.15</u> $S_{max}'^{z}$, total deficiency at the beginning of subproblem z, $z \geq 1$ is bounded by $3|V|$.

<u>Proof</u>: Note that $S_{max}'^{z} = \sum_{\alpha\in Q^{z}} (S_{max,\alpha})^{B}$

For $i,j\in Q^{z}$, we have $V_i' \cap V_j' = \emptyset$. Thus, using Lemma 2.14, we have

$$S_{max}'^{z} \leq 3|V|. \blacksquare$$

Theorem 2.5. Total number of pivots on the new scaling algorithm, DUALSC is bounded by $3(\tau+1)|V|^2$. Moreover, the algorithm requires $O[(\tau+1)|V|^3]$ arithmetic operations.

Proof: Same as proof of Theorem 2.4.

The bound we have obtained for the algorithm DUALSC depends on the logarithm of the total supplies (or demands), thus DUALSC is not a genuinely polynomial algorithm. However, when we compare this bound with the computational bounds of the other dual simplex algorithms in the literature, we see that it is an $O(|V|^2)$ better than Ikura and Nemhauser's algorithm (designed for transportation problems only). Orlin's two genuinely polynomial dual simplex algorithms have computational bounds of $O(|V^6|\log|V|)$ and $O(|V^4|\log|V|)$ when $BIT(b) = \infty$. We want to restate here that $BIT(b)$ is the minimum value of s such that $2^s b$ is integer valued, and b is the vector of supply and demand values with $|b_i| \leq 1$. We believe that in most cases $BIT(b)$ will be equal to infinity. Thus algorithm DUALSC compares very favorably with all the other dual simplex algorithms. To compare our algorithm empirically, we conducted a computational experimentation which compares all the dual simplex algorithms on randomly generated problems. We present this experimentation in Section 2.8.

2.7 A Heuristic Leaving-Arc Selection Rule

Before conducting our computational experience, we introduce another dual simplex algorithm that differs from DUAL described in Section 2.4.5 in the determination of the leaving arc.

In this algorithm, step 1a. of algorithm DUAL is changed as follows :

Step 1a$'$)

Let $Q = \{ i : i \in V_F,\ S_i > 0 \}$

If $p' = 0$ THEN

Find T_p such that $D(p) = \max\limits_{j \in Q} \{D(j)\}$.

ELSE

Find T_p such that $D(p) = \max\limits_{j \in Q \cap P_p'} \{D(j)\}$

ENDIF

We will call this algorithm MAXDEP since the node to be pivoted on is the one with the maximum depth among all the possible candidates. This selection rule corresponds to deleting the first found arc with a negative flow. Even though we could not obtain a polynomial bound for this algorithm, we will include it in the computational comparisons since MAXDEP avoids the computation of the S_i' values and may require less computation time.

2.8. Computational Experience

Our purpose is to compare the algorithms DUAL and DUALSC given in Sections 2.4.5 and 2.6 respectively with other dual simplex algorithms available in the literature and with a primal simplex algorithm as well as the algorithm MAXDEP. Two performance measurements will be considered; number of simplex (primal or dual) pivots and the CPU time.

The only two dual simplex algorithms for minimum cost flow problem with polynomial bounds in the literature are given by Orlin [29]. We included them in our computational comparison and called them ORLIN1 and ORLIN2. The primal simplex algorithm that we included in our study is the GNET program [16], a readily available state-of-the-art code.

In the algorithms DUAL and DUALSC, we used three node-length arrays to represent the spanning tree. These are the predecessor, the thread and the last node in the subtree indices. Three node-length arrays are used to store the dual values, and the S_i and S_i' values. For the algorithm DUALSC, two more node-length arrays are used to store the S_i and S_i' values for the final subproblem. Three arc-length arrays are used to store the input data. They are from-node, to-node and cost. Two node-length arrays are used as pointers to from-node and to-node arrays. Weights of nodes are stored in one node-length array. We also used other node-length arrays (to store the inverse thread, a flag to indicate the nodes in a given subtree, and to store the orientation of the arcs) to reduce the computational time further. We found that these additional arrays reduced the overall computation time even though extra time was spent to update them.

The coding of ORLIN1 and ORLIN2 are straightforward implementations of the algorithms. In these algorithms, the spanning tree is represented with three node-length arrays; the predecessor, the thread and the depth indices.

We now summarize the memory requirements of all each code.

Table 2.1: Memory Requirements for each of the Codes.

Code	Memory Locations Used
DUAL	$13\|V\| + 3\|A\|$
DUALSC	$15\|V\| + 3\|A\|$
GNET	$9\|V\| + 3\|A\|$
MAXDEP	$12\|V\| + 3\|A\|$
ORLIN1	$13\|V\| + 3\|A\|$
ORLIN2	$13\|V\| + 3\|A\|$

All of the algorithms tested are coded in FORTRAN 77 and are run on a VAX 11/750 using the VAX-11 Fortran V3.0 compiler. All runs are made overnight when we were the only users of the system. All algorithms are compared on the same set of test problems. Test problems are randomly generated by the NETGEN program developed in [36]. For minimum cost flow problems, the input to NETGEN problem are the following; number of source nodes, number of demand nodes, number of transshipment source nodes, number of transshipment demand nodes, number of arcs, cost range and total supply (or demand). Here

transshipment source (demand) nodes are those source (demand) nodes into (from) which arcs are allowed to lead (emanate).

The computational studies are carried out in two phases. In the first phase, the algorithms are compared for small problems. The characteristics of the problems generated in phase one are given in Table 2.2.

Table 2.2: Characteristics of the Problems Generated in Phase 1.

Problem Type	No. of Nodes	No. of Source Nodes	No. of Demand Nodes	No. of Trans. Source Nodes	No. of Trans. Demand Nodes	No. of Arcs	Total Supply	Cost Range
M-1	50	5	5	5	5	2000	10000	1-100
M-2	50	5	5	5	5	1500	10000	1-100
M-3	50	5	5	5	5	750	10000	1-100
M-4	75	5	9	5	9	4500	20000	1-100
M-5	75	5	9	5	9	3375	20000	1-100
M-6	75	5	9	5	9	1688	20000	1-100
M-7	100	10	20	5	5	8000	30000	1-100
M-8	100	10	20	5	5	6000	30000	1-100
M-9	100	10	20	5	5	3000	30000	1-100

Note that problems M-1, M-4, and M-7 are dense problems whereas M-3, M-6 and M-9 are sparse problems.

Results of the experiments in phase one are summarized in Table 2.3 where the mean values are obtained by averaging ten runs.

The CPU times that are reported are in seconds, and they do not include the input-output time. The time statistics are collected using the library routines LIB$INIT_TIMER and LIB$STAT_TIMER.

Table 2.3. The Performance of Algorithms in Phase 1

	No. of Pivots Min	Max	Mean	CPU Times Min	Max	Mean
Problem M-1						
DUAL	12	33	23.8	0.29	0.48	0.43
DUALSC	15	34	25.5	0.35	0.59	0.52
GNET	63	83	74.3	0.38	0.58	0.51
MAXDEP	20	60	42.3	0.28	0.51	0.47
ORLIN1	14	38	25.7	0.99	1.28	1.18
ORLIN2	16	39	26.1	1.01	1.32	1.20
Problem M-2						
DUAL	10	30	21.9	0.20	0.39	0.31
DUALSC	10	34	22.4	0.27	0.57	0.41
GNET	57	91	76.0	0.28	0.48	0.40
MAXDEP	19	42	27.9	0.21	0.46	0.33
ORLIN1	13	35	23.4	0.99	1.20	1.11
ORLIN2	17	36	24.1	1.01	1.23	1.14
Problem M-3						
DUAL	10	31	18.4	0.14	0.27	0.18
DUALSC	10	33	23.0	0.21	0.40	0.31
GNET	65	94	75.4	0.21	0.31	0.25
MAXDEP	11	31	23.8	0.12	0.25	0.19
ORLIN1	10	33	19.2	0.59	0.73	0.65
ORLIN2	13	35	21.2	0.60	0.77	0.67
Problem M-4						
DUAL	33	43	36.8	0.87	1.42	1.11
DUALSC	32	43	39.2	0.91	1.62	1.32
GNET	115	146	129.3	1.15	1.65	1.39
MAXDEP	34	56	45.5	0.79	1.39	1.09
ORLIN1	35	45	38.4	2.98	3.45	3.12
ORLIN2	40	48	42.5	3.05	3.62	3.27

Table 2.3. (continued)

	No. of Pivots			CPU Times		
	Min	Max	Mean	Min	Max	Mean
Problem M-5						
DUAL	25	51	35.0	0.63	1.03	0.84
DUALSC	30	42	34.0	0.69	1.06	0.92
GNET	112	137	123.6	0.90	1.15	1.03
MAXDEP	36	68	47.6	0.55	1.18	0.89
ORLIN1	27	54	38.7	2.51	3.03	2.87
ORLIN2	30	61	42.1	2.48	3.13	2.91
Problem M-6						
DUAL	18	34	26.8	0.38	0.46	0.43
DUALSC	21	40	33.5	0.45	0.87	0.71
GNET	91	131	117.3	0.45	0.68	0.58
MAXDEP	28	34	30.5	0.37	0.50	0.44
ORLIN1	21	39	30.2	1.44	1.98	1.70
ORLIN2	24	45	34.1	1.65	1.89	1.79
Problem M-7						
DUAL	40	59	49.8	1.71	2.69	2.13
DUALSC	35	60	50.0	1.94	3.14	2.66
GNET	150	182	168.5	2.18	2.99	2.68
MAXDEP	41	76	61.5	1.63	3.33	2.45
ORLIN1	42	65	52.1	3.99	5.03	4.75
ORLIN2	46	71	58.0	4.02	5.14	4.84
Problem M-8						
DUAL	32	69	47.3	1.30	2.12	1.60
DUALSC	39	83	59.1	1.48	3.47	2.29
GNET	151	176	158.1	1.74	2.39	2.01
MAXDEP	49	130	80.6	1.25	2.98	2.17
ORLIN1	28	74	50.2	2.94	4.76	3.91
ORLIN2	31	77	53.5	2.97	4.87	3.93
Problem M-9						
DUAL	28	64	44.4	0.55	1.28	0.89
DUALSC	29	68	49.0	0.77	1.52	1.13
GNET	147	174	164.6	1.16	1.27	1.20
MAXDEP	43	117	68.4	0.55	1.62	0.99
ORLIN1	30	69	49.2	3.12	3.99	3.69
ORLIN2	32	73	51.2	3.34	3.98	3.74

In almost all but one of the nine problem types, algorithm DUAL was faster than all the other algorithms tested. When we examine the ratios of CPU times of other algorithms to the CPU times of algorithm DUAL in Table 2.4, we see that DUAL is two to four times faster than algorithms ORLIN1 and ORLIN2. Algorithm DUALSC is faster in dense problems and slower in sparse problems. Algorithm MAXDEP is almost as fast as algorithm DUAL. In problem type M-4, it is even faster. However, in problem type M-8, it is relatively slower than DUAL. In this problem type, MAXDEP required almost twice the pivots algorithm DUAL required.

Next, we will examine the CPU time per pivot for each of the algorithms in Table 2.5.

Algorithm GNET requires three to four times more pivots compared to the other algorithms. However, it requires the minimum amount of time per pivot.

Table 2.4. Ratio of the CPU Times of Algorithms to the CPU Time of Algorithm DUAL

Problem Type	DUALSC	GNET	MAXDEP	ORLIN1	ORLIN2
M-1	1.21	1.19	1.09	2.74	2.79
M-2	1.32	1.29	1.06	3.58	3.68
M-3	1.72	1.38	1.06	3.61	3.72
M-4	1.19	1.25	0.98	2.81	2.95
M-5	1.09	1.23	1.06	3.41	3.46
M-6	1.65	1.35	1.02	3.95	4.16
M-7	1.24	1.26	1.15	2.22	2.27
M-8	1.43	1.26	1.36	3.21	3.24
M-9	1.27	1.35	1.11	4.15	4.20

Table 2.5. Time per Pivot for each of the Algorithms

Problem Type	DUAL	DUALSC	GNET	MAXDEP	ORLIN1	ORLIN2
M-1	.018	.021	.007	.011	.046	.046
M-2	.014	.018	.005	.012	.047	.047
M-3	.010	.013	.003	.008	.033	.031
M-4	.030	.034	.011	.024	.081	.076
M-5	.024	.027	.008	.019	.074	.069
M-6	.016	.021	.005	.014	.056	.052
M-7	.043	.053	.016	.039	.085	.083
M-8	.034	.039	.013	.027	.078	.073
M-9	.020	.023	.007	.014	.075	.073

In phase two of the experimentation, we will compare the algorithms for larger problems. However, since the performance of algorithms ORLIN1 and ORLIN2 are very similar and poorer compared to the other algorithms for small problems, we will include only the ORLIN1 algorithm in the second phase.

In phase two, we generated two groups of test problems, one dense and the other sparse. The characteristics of these problems are summarized in Table 2.6.

Table 2.6. Characteristics of the Problems Generated in Phase 2

Problem Type	No. of Nodes	No. of Source Nodes	No. of Demand Nodes	No. of Trans. Source Nodes	No. of Trans. Demand Nodes	No. of Arcs	Total Supply	Cost Range
MCD-1	150	20	40	10	20	18000	10000	1-1000
MCD-2	150	20	50	10	30	16000	20000	1-1000
MCD-3	200	10	50	10	30	28000	30000	1-1000
MCD-4	200	10	20	5	20	32000	30000	1-1000
MCS-1	1000	70	50	30	20	10000	10000	1-1000
MCS-2	1500	100	100	50	50	22500	20000	1-1000
MCS-3	2000	200	200	90	90	30000	30000	1-1000

Problems MCD-1 to MCD-4 are dense problems whereas MCS-1 to MCS-3 are sparse problems. Results of the experiments in phase two are summarized in Table 2.7.

Table 2.7. Performance of Algorithms in Phase Two

	No. of Pivots Min	Max	Mean	CPU Times Min	Max	Mean
Problem MCD-1						
DUAL	89	110	96.7	6.44	9.15	7.75
DUALSC	88	101	94.2	6.28	10.26	7.84
GNET	244	277	259.0	6.48	7.28	7.02
MAXDEP	128	148	139.5	6.69	8.35	7.36
ORLIN1	95	130	102.5	15.11	24.71	20.16
Problem MCD-2						
DUAL	87	115	98.1	6.56	8.43	7.12
DUALSC	86	121	103.4	6.68	9.38	7.25
GNET	260	294	275.2	6.15	8.13	6.99
MAXDEP	161	185	172.4	6.99	12.66	9.65
ORLIN1	93	116	103.5	13.42	22.13	19.15
Problem MCD-3						
DUAL	46	89	72.1	7.36	13.15	10.54
DUALSC	63	110	87.1	9.09	17.74	13.23
GNET	306	380	344.5	9.81	12.59	11.21
MAXDEP	90	114	107.3	9.54	15.88	12.58
ORLIN1	51	94	73.2	25.13	32.71	29.62
Problem MCD-4						
DUAL	81	143	125.2	12.15	23.27	16.11
DUALSC	98	186	138.1	13.76	32.39	19.49
GNET	316	390	352.5	11.47	16.87	14.12
MAXDEP	161	292	250.4	14.67	27.88	21.23
ORLIN1	98	146	126.2	30.12	71.34	53.24

Table 2.7 (continued)

	No. of Pivots Min	Max	Mean	CPU Times Min	Max	Mean
Problem MCS-1						
DUAL	768	968	840.2	44.07	60.31	54.12
DUALSC	804	1248	934.8	63.76	90.12	75.34
GNET	2100	2314	2218.9	24.16	26.14	25.76
MAXDEP	2661	3212	2812.5	42.12	60.12	50.70
ORLIN1	2010	2157	2098.1	224.60	255.16	250.91
Problem MCS-2						
DUAL	3100	3321	3214.9	90.60	162.44	134.23
DUALSC	3134	3414	3297.1	130.36	175.45	157.39
GNET	3301	3547	3421.6	40.31	46.12	43.60
MAXDEP	4276	5412	4892.2	75.45	151.31	110.25
ORLIN1	3623	3817	3756.1	403.18	501.90	489.10
Problem MCS-3						
DUAL	4215	4515	4412.1	200.45	250.20	244.17
DUALSC	4314	4575	4476.8	278.56	302.71	295.16
GNET	4612	4819	4798.0	80.16	85.15	83.16
MAXDEP	5513	5812	5676.2	215.23	254.65	223.78
ORLIN1	4278	4565	4423.4	696.52	787.82	723.02

Within the size range of problems tested, we can see that for dense problems, algorithms DUAL,DUALSC, MAXDEP and GNET are very competitive. In particular, when we compare algorithms DUAL and GNET, we see that for problems MCD-1, MCD-2, and MCD-4, GNET is slightly faster than DUAL whereas for problem type MCD-3, DUAL is slightly faster than GNET. When we compare them in terms of number of pivots criterion, we see that GNET required three to five times more pivots than DUAL for dense problems.Algorithm MAXDEP performed as good as DUAL and GNET in the first three dense problems.

Algorithm ORLIN1 performed poorer compared to the other algorithms for all the problem types in phase two. It required three to five times more CPU time.

When we examine the performance of algorithms for sparse problems, we see that GNET was superior compared to all the other algorithms. It is twice or three times as fast as algorithm DUAL. For problem types MCS-2 and MCS-3, the number of pivots GNET required and the number of pivots DUAL required are close to each other. In these problems, GNET is three times faster than DUAL.

Based on the experimentation, we can conclude that algorithm DUAL is faster than GNET for small problems. For large, dense problems, they are competitive. However, for large, sparse problems, algorithm GNET is two to three times faster than algorithm DUAL.

Now, we will compare the results obtained from this study to the results given by Glover, Karney and Klingman [10]. In their study, they compared the primal-simplex algorithm PNET with other algorithms, mainly with a dual-simplex algorithm DNET, and out-of-kilter algorithms. Their conclusion was PNET was superior to all the other algorithms. In particular, their results showed that PNET is 6-12 times faster than DNET. There is a major difference between their conclusion and ours on performance of the dual codes. The basic reasons for this gap can be summarized in the following way :

a) In DNET, the leaving arc is determined by the first negative criterion. In DUAL and DUALSC , the leaving arc selection rules are different.

b) They only tested for sparse problems.

c) As they have concluded in their study, the representation of the basis in DNET is not the most efficient representation for the dual approach. Using the inverse-thread index reduces the CPU time a great deal in DUAL and DUALSC.

CHAPTER 3

THE TRANSPORTATION PROBLEM

3.1 Formulation of the Problem

The transportation problem is a special case of the minimum cost flow problem. Consider the network representation $G = (V,E)$ rooted at node 0 as discussed in section 2.3 with $V = V_S \cup V_D$. If there exists a direct arc from $i \in V_S$ to $j \in V_D$, then $(i,j) \in E$. For each arc $(i,j) \in E$, we associate a nonnegative integer c_{ij} which corresponds to a unit cost over the arc (i,j). For each node $i \in V_S$, we associate a positive integer a_i designating the supply at that point and for each node $i \in V_D$, we associate a positive integer b_j corresponding to the demand at that point.

Then, we can formulate the transportation problem (TP) exactly the same as the (MCP) formulation.

$$\text{(TP)} \quad \text{Minimize} \sum_{(i,j) \in E} c_{ij} x_{ij} \tag{3.1}$$

subject to

$$\sum_{(i,j) \in \alpha(i)} x_{ij} - \sum_{(i,j) \in \beta(i)} x_{ij} = \begin{cases} a_i & \text{if } i \in V_S \\ -b_i & \text{if } i \in V_D \end{cases} \tag{3.2}$$

$$x_{ij} \geq 0 \qquad (i,j) \in E \quad . \tag{3.3}$$

3.2 The Dual Simplex Algorithms for the Transportation Problem

The steps of algorithms DUAL, described in Section 2.5.5 and DUALSC, described in Section 2.6 can be applied directly for the transportation problem. However, the construction of the initial tree can be changed to take advantage of the special structure of the transportation problem. In this section, we will only restate the construction of the initial tree, that is the initialization step.

Let T_0' be a shortest path tree of G rooted at node 0 using costs as the weights. Such a tree can easily be constructed by attaching each supply node to node 0 and attaching each demand node j to a supply node x with $\min\{c_{ij} : i \in V_S\} = c_{xj}$. Let w_i represent the negative of the shortest path from node 0 to node i to T_0'. Now attach each source node i, which is not attached to a demand node, to a demand node k (after deleting arc (0,i)) with $w_k + c_{ik} = \min \{ w_j + c_{ij} : j \in V_D\}$ and set $w_i = w_k + c_{ik}$.

3.3 Computational Experience

In this section, we will compare algorithms DUAL and DUALSC with the algorithms discussed in section 2.7 on transportation problems. Moreover, we will also include the unscaled and scaled versions, named DTRAN and DTRANSC respectively, of the dual simplex algorithm developed by Ikura and Nemhauser [32,33]. The codings of DTRAN and DTRANSC are the original ones obtained from the autors.

Again, all of the algorithms tested are coded in FORTRAN 77 and are run on a VAX 11/750 using the VAX-11 Fortran V3.0 compiler. All

runs are made overnight when we were the only users of the system. All algorithms are compared on the same set of test problems. Test problems are generated by the NETGEN program.

The computational studies are carried out in two phases. In the first phase, the algorithms are compared for small sized problems. The characteristics of the problems generated in phase one are given in Table 3.1.

Table 3.1 Characteristics of the Problems Generated in Phase One

Problem Type	Number of Source Nodes	Number of Demand Nodes	Number of Arcs	Total Supply	Cost Range
T-1	70	70	4900	10000	1-100
T-2	70	70	2940	10000	1-100
T-3	70	70	1470	10000	1-100
T-4	50	90	4500	10000	1-100
T-5	50	90	2700	10000	1-100
T-6	50	90	1350	10000	1-100
T-7	90	50	4500	10000	1-100
T-8	90	50	2700	10000	1-100
T-9	90	50	1350	10000	1-100

Note that problems T-1, T-4, and T-7 are dense problems in which the density equals one. Here we define the density as the ratio of the number of actual arcs in the transportation network to the maximum possible number of arcs. Problems T-3, T-6 and T-9 are sparse problems in which the density equals 0.3. For problems T-2, T-5 and T-8, the density equals 0.6.

Results of the experiments in phase one are summarized in Table 2.3 where the mean values are obtained by averaging ten runs.

The CPU times that are reported are in seconds, and they do not

include the input-output time. The time statistics are collected using the library routines LIB$INIT_TIMER and LIB$STAT_TIMER.

Table 3.2 The Performance of the Algorithms in Phase One.

	No. of Pivots			CPU Times		
	Min	Max	Mean	Min	Max	Mean
Problem T-1						
DUAL	166	223	189.8	2.03	3.54	2.84
DUALSC	172	224	199.8	3.43	5.42	4.32
DTRAN	198	250	224.0	3.54	5.31	4.42
DTRANSC	245	276	254.0	3.72	6.28	4.58
ORLIN1	190	223	207.0	6.01	7.44	6.55
ORLIN2	214	259	230.0	6.09	8.34	7.33
GNET	231	250	239.0	3.01	3.46	3.23
MAXDEP	166	358	251.6	2.30	3.94	2.86
Problem T-2						
DUAL	173	189	180.8	1.52	2.04	1.77
DUALSC	186	209	196.8	2.67	3.77	3.05
DTRAN	213	257	233.6	2.91	4.77	3.72
DTRANSC	250	280	263.6	2.84	3.43	3.20
ORLIN1	206	245	227.8	6.23	7.82	6.73
ORLIN2	223	286	257.8	6.53	8.51	7.21
GNET	251	292	275.2	2.32	2.86	2.60
MAXDEP	199	310	247.4	1.61	2.61	1.92
Problem T-3						
DUAL	172	190	182.0	1.08	1.33	1.21
DUALSC	177	212	189.8	1.93	2.35	2.07
DTRAN	212	237	224.3	1.93	2.72	2.23
DTRANSC	226	280	252.0	1.52	2.50	2.01
ORLIN1	185	247	217.5	4.19	6.65	5.22
ORLIN2	216	282	246.0	4.49	6.46	5.44
GNET	239	273	250.5	1.41	1.63	1.54
MAXDEP	231	254	239.3	1.03	1.27	1.15
Problem T-4						
DUAL	135	173	157.0	1.79	2.99	2.48
DUALSC	139	187	158.0	2.64	3.97	3.17
DTRAN	211	229	220.4	3.32	4.29	3.87
DTRANSC	239	270	254.8	3.41	5.88	4.17
ORLIN1	166	200	178.0	5.16	7.00	6.02
ORLIN2	183	236	202.3	5.84	7.45	6.38
GNET	234	255	241.4	2.79	3.23	2.98
MAXDEP	197	287	213.8	1.81	2.69	2.27

Table 3.2 (continued)

	No. of Pivots Min	Max	Mean	CPU Times Min	Max	Mean
Problem T-5						
DUAL	142	191	170.4	1.27	1.96	1.76
DUALSC	171	210	182.3	2.48	2.90	2.78
DTRAN	201	257	235.0	2.53	4.74	3.87
DTRANSC	206	255	228.4	1.67	2.98	2.57
ORLIN1	209	262	234.6	6.49	7.88	6.94
ORLIN2	228	298	258.2	5.78	8.03	7.04
GNET	219	295	260.8	2.14	2.62	2.40
MAXDEP	173	287	234.0	1.06	2.24	1.76
Problem T-6						
DUAL	141	183	166.6	0.98	1.35	1.13
DUALSC	174	191	181.6	1.87	2.21	1.62
DTRAN	235	260	246.0	2.23	3.16	2.84
DTRANSC	211	270	248.8	1.59	2.69	2.18
ORLIN1	189	211	202.8	4.53	5.21	4.86
ORLIN2	208	255	235.3	4.45	5.66	4.99
GNET	249	281	270.0	1.52	1.75	1.63
MAXDEP	180	283	232.0	0.83	1.45	1.10
Problem T-7						
DUAL	174	204	192.8	2.04	2.90	2.50
DUALSC	181	216	197.4	2.75	4.02	3.49
DTRAN	204	234	221.0	3.27	5.11	4.05
DTRANSC	222	246	231.6	2.43	4.14	3.52
ORLIN1	183	214	203.2	5.39	6.87	6.28
ORLIN2	187	212	204.5	5.41	6.72	6.23
GNET	224	286	244.8	2.87	3.99	3.35
MAXDEP	240	288	262.2	1.85	3.35	2.77
Problem T-8						
DUAL	163	219	187.0	1.18	2.69	1.91
DUALSC	165	224	191.5	2.40	3.08	2.66
DTRAN	198	229	213.0	2.65	3.41	2.96
DTRANSC	211	232	221.8	1.80	3.16	2.52
ORLIN1	197	232	226.0	5.97	8.23	6.84
ORLIN2	227	283	258.8	6.21	8.08	6.88
GNET	242	265	250.5	2.34	2.56	2.46
MAXDEP	195	314	254.5	1.26	3.04	2.15

Table 3.2. (continued)

	No. of Pivots			CPU Times		
	Min	Max	Mean	Min	Max	Mean
Problem T-9						
DUAL	160	199	177.8	0.84	1.38	1.14
DUALSC	166	206	181.3	1.42	2.13	1.91
DTRAN	197	208	202.0	1.61	1.82	1.71
DTRANSC	211	234	227.0	1.44	2.04	1.75
ORLIN1	190	258	223.3	4.39	5.96	5.30
ORLIN2	227	280	259.0	4.09	7.46	5.97
GNET	216	251	233.0	1.38	1.64	1.50
MAXDEP	204	294	241.8	0.90	1.79	1.29

In all but three of the nine problem types, algorithm DUAL was faster than all the other algorithms tested. When we examine the ratios of CPU times of other algorithms to the CPU times of algorithm DUAL in Table 3.3, we see that DUAL is three to five times faster than algorithms ORLIN1 and ORLIN2. Algorithm DUALSC is faster in dense problems and slower in sparse problems whereas GNET is faster in sparse problems and slower in denser problems. Algorithm MAXDEP is as fast as DUAL and in some cases faster (Problems M-5, M-6, and M-9).

When we compare algorithms DTRAN and DTRANSC, we see that DTRANSC runs faster than DTRAN in most of the problem types even though it required more iterations. This observance agrees with the conclusion in [33]. When we compare DUALSC and DTRANSC we see that there is no major dominance of one over the other. In some problems DUALSC performed better, and in others DTRANSC's performance was better. However, there is

a observable dominance of DUALSC over DTRAN. DUALSC is 20-50 percent faster than DTRAN.

Table 3.3. Ratio of the CPU Times of Algorithms to the CPU Time of Algorithm DUAL.

	DUALSC	DTRAN	DTRANSC	ORLIN1	ORLIN2	GNET	MAXDEP
T-1	1.52	1.55	1.61	2.31	2.58	1.14	1.01
T-2	1.72	2.10	1.81	3.80	4.07	1.47	1.08
T-3	1.71	1.84	1.66	4.31	4.50	1.27	0.95
T-4	1.28	1.56	1.68	2.43	2.57	1.20	0.91
T-5	1.57	2.20	1.46	3.94	4.00	1.36	1.00
T-6	1.43	2.51	1.93	4.30	4.42	1.44	0.97
T-7	1.40	1.62	1.41	2.51	2.75	1.34	1.11
T-8	1.39	1.55	1.32	3.59	3.60	1.29	1.13
T-9	1.57	1.50	1.53	4.65	5.24	1.32	1.13

Next, we will examine the CPU time per pivot for each of the algorithms in Table 3.4.

Algorithms DUAL, MAXDEP, and GNET require the minimum amount of time per pivot. Algorithm DUALSC requires almost twice the time per pivot of algorithm DUAL. We could expect this since in algorithm DUALSC, for determining the leaving arc, we need to determine S_i' values for the current subproblem and the final subproblem.

Table 3.4. Time per Pivot for each of the Algorithms

	DUAL	DUALSC	DTRAN	DTRANSC	ORLIN1	ORLIN2	GNET	MAXDEP
T-1	0.015	0.021	0.019	0.018	0.031	0.032	0.013	0.011
T-2	0.010	0.015	0.016	0.012	0.030	0.028	0.009	0.008
T-3	0.007	0.011	0.010	0.008	0.024	0.022	0.006	0.005
T-4	0.016	0.020	0.018	0.016	0.034	0.032	0.012	0.011
T-5	0.010	0.015	0.016	0.011	0.030	0.027	0.009	0.008
T-6	0.007	0.009	0.012	0.009	0.024	0.021	0.006	0.005
T-7	0.013	0.018	0.018	0.015	0.033	0.031	0.014	0.011
T-8	0.010	0.014	0.014	0.011	0.030	0.027	0.010	0.008
T-9	0.006	0.011	0.008	0.008	0.024	0.023	0.006	0.005

In phase two of the experimentation, we will compare the algorithms for larger problems. However, since algorithms ORLIN1 and ORLIN2 performed poorer compared to the others in small sized problems, we will include only ORLIN1 in the second phase. In phase two, we generated two groups of test problems, one dense and the other sparse. The characteristics of these problems are summarizes in Table 3.5.

Table 3.5. Characteristics of the Problems Generated in Phase 2.

Problem Type	Number of Source Nodes	Number of Demand Nodes	Number of Arcs	Total Supply	Cost Range
TD-1	100	200	16000	10000	1-1000
TD-2	150	150	18000	20000	1-1000
TD-3	150	200	24000	25000	1-1000
TD-4	200	200	32000	30000	1-1000
TS-1	150	150	2250	10000	1-1000
TS-2	500	1000	5000	10000	1-1000
TS-3	1000	1000	10000	20000	1-1000
TS-4	1500	1000	15000	30000	1-1000

Results of the experiments in phase two are summarized in Table 3.6.

Table 3.6. The Performance of the Algorithms in Phase Two.

	No. of Pivots			CPU Times		
	Min	Max	Mean	Min	Max	Mean
Problem TD-1						
DUAL	369	451	407.0	12.68	17.87	15.10
DUALSC	377	449	420.3	19.24	23.56	20.71
MAXDEP	567	743	639.0	14.83	19.18	16.76
DTRANSC	470	646	556.5	21.89	39.58	30.71
GNET	549	717	618.5	12.37	16.50	14.17
ORLIN1	381	492	455.1	31.12	42.65	38.72
Problem TD-2						
DUAL	380	532	433.0	13.16	19.67	15.63
DUALSC	387	547	469.1	18.68	25.21	20.81
MAXDEP	570	722	652.0	14.73	22.06	17.60
DTRANSC	507	627	554.4	18.87	32.14	25.71
GNET	527	630	604.2	13.54	17.36	15.84
ORLIN1	411	562	477.1	42.16	50.43	47.90
Problem TD-3						
DUAL	462	545	514.8	20.04	27.38	23.53
DUALSC	476	548	520.5	39.12	44.39	40.29
MAXDEP	640	796	727.2	19.49	31.13	24.38
DTRANSC	688	851	735.2	49.78	120.64	74.25
GNET	712	851	761.5	21.22	27.02	23.66
ORLIN1	500	592	567.0	80.12	91.78	88.31
Problem TD-4						
DUAL	554	631	600.1	28.71	38.16	34.91
DUALSC	579	681	652.7	49.27	64.12	59.39
MAXDEP	799	1797	1193.4	36.94	132.40	64.51
DTRANSC	744	844	804.6	64.11	119.72	86.80
GNET	839	928	870.1	31.92	34.71	32.12
ORLIN1	636	712	685.1	91.34	102.01	98.91

Table 3.6 (continued)

	No. of Pivots Min	Max	Mean	CPU Times Min	Max	Mean
Problem TS-1						
DUAL	363	477	410.6	3.25	4.30	3.63
DUALSC	401	490	450.2	4.34	5.28	4.78
MAXDEP	506	676	570.6	2.82	4.02	3.55
DTRANSC	535	597	568.1	7.19	7.87	7.43
GNET	515	624	554.0	3.29	4.01	3.59
ORLIN1	412	499	465.1	8.11	8.98	8.65
Problem TS-2						
DUAL	1825	1965	1894.3	43.22	58.07	50.60
DUALSC	1901	2003	1957.2	68.23	79.49	70.32
MAXDEP	2661	3030	2406.9	36.54	55.69	44.64
DTRANSC	2022	2323	2196.1	65.10	81.45	74.80
GNET	2173	2319	2210.1	22.08	25.17	23.43
ORLIN1	2145	2218	2193.0	214.19	245.90	224.18
Problem TS-3						
DUAL	3126	3328	3180.5	90.60	122.92	105.51
DUALSC	3176	3341	3190.1	150.31	200.35	178.31
MAXDEP	4260	5646	4892.5	75.88	151.30	106.33
DTRANSC	3233	3612	3451.3	165.15	220.27	195.23
GNET	3305	3649	3449.1	43.31	47.40	45.37
ORLIN1	3712	3908	3892.2	411.55	499.43	455.89
Problem TS-4						
DUAL	4167	4452	4392.1	170.12	190.23	183.26
DUALSC	4198	4513	4476.9	240.54	290.53	275.39
MAXDEP	5412	6587	5912.7	160.40	195.76	180.65
DTRANSC	4243	4816	4698.3	280.32	320.23	302.88
GNET	4456	4787	4648.1	70.25	78.65	72.88
ORLIN1	4423	4654	4532.3	675.12	712.48	699.89

When we examine Table 3.6, we can see that for dense problems, algorithms DUAL and GNET are very competitive within the size range of

problems tested. Algorithm MAXDEP performed as good as DUAL and GNET in the first three dense problems, however, for problem T-4, DUAL and GNET were almost twice as fast as MAXDEP. When we compare algorithms DUAL and ORLIN1, we can see that DUAL is almost three times faster than ORLIN1 for dense problems. Also, DUAL is almost twice as fast as DTRANSC.

When we examine the performance of algorithms for sparse problems, we see that GNET was superior compared to all the other algorithms for larger problems. It is twice as fast as algorithm DUAL. However, for problem TS-1, again DUAL and GNET are competitive.

In conlusion, algorithm DUAL is faster than all the other dual simplex algorithms in the literature for all the problem types considered. It is competitive with GNET for large dense problems, however, for large sparse problems GNET is 2-2.5 times faster than DUAL.

These results agree with our conclusion for the minimum cost flow problems given in Section 2.8.

CHAPTER 4
THE ASSIGNMENT PROBLEM

Assignment problems arise when a number of (jobs) should be assigned to a number of (machines) with the minimum cost of assignment.

4.1. The Problem Formulation

The assignment problem is a special case of the transportation problem. In the (TP) formulation in Section 3.1, if $a_i = 1$ for all $i \in V_S$, and $b_j = 1$, for all $j \in V_D$, and $|V_S| = |V_D| = N$, then the problem is called an assignment problem (AP).

4.2 Literature Review

For assignment problems, several algorithms with polynomial bounds have been developed.

First algorithm developed for assignment problems is Kuhn's [37] "Hungarian Method" with a computational bound of $O(N^3)$ operations.

Barr, Glover and Klingman's [20] (AP-AB) (Alternating Path - Alternating Basis) primal simplex algorithm specialized for assignment problems gave good results computationally, but no theoretical bound is known for this algorithm.

Hung and Rom [38] developed a recursive algorithm based on relaxing the given problem which exploits the (AP - AB) structure. The algorithm has a computational bound of $O(N^3)$ operations.

Engquist's [39] algorithm, which is similar to Hung and Rom's algorithm, solves the assignment problem by solving successive shortest path problems. The bound of this algorithm is $O(N^3)$ operations.

Hung [40] developed a primal simplex algorithm for the assignment problem which generates at most $N^3 \ln \Delta$ bases where Δ is the difference in the objective value between an initial extreme point and the optimal extreme point. While preserving (AP - AB) structure, the algorithm provides a polynomial bound in the problem size and the encoding of the problem data by applying the "Modified Row Most Negative" rule in selecting the entering arc.

Bertsekas's [41] algorithm for the assignment problem is a primal-dual algorithm which he calls as an "outpricing" algorithm with a complexity of $O(N^3) + O(RN^2)$ where R is an integer valued number greater than all cost entries. For large R, the algorithm is modified by combining with the Hungarian method to give a worst case complexity of $O(V^3)$.

In 1985, Balinski [42] provided a dual simplex algorithm for the assignment problem that terminates in at most $(N-1)(N-2)/2$ pivots. The algorithm is guided by the signatures, where signature of a tree T is the vector of its source node degrees. The objective of the method is to find a tree whose signature contains exactly one 1 and otherwise 2's. The worst case bound of this algorithm agrees with the bound of our algorithm. The main difference between Balinski's algorithm and the

DUAL algorithm is that in his algorithm, one can select a subtree T_i, $i \in V_S$ for pivoting while $S_i'=0$. Algorithm DUAL requires $S_i' > 0$ for T_i to be selected for pivoting. Due to this difference, Balinski's algorithm cannot detect a dual feasible basis which admits an optimal assignment. However, in 1986, Balinski [43] provided a competitive dual simplex algorithm with the same computational bound as the one in [42], which requires S_i' to be greater than zero if T_i is to be selected for pivoting. Hence, this new algorithm is equivalent to DUAL applied to assignment problems. However, Balinski's algorithm cannot be extended to solve the minimum cost flow problem or the transportation problem.

4.3 A Dual Simplex Algorithm for the Assignment Problem and its Computational Bound

The dual simplex algorithm is exactly the same as the one given in Section 3.2.

The initial total deficiency as well as total deficiency during the course of the algorithm is defined in the same manner as in the minimum cost flow problem, i.e.

$$S_{max} = \sum_{i \in Q} S_i' \qquad (3.1)$$

Note that S_{max} is equivalent to the number of unsuccessful assignments in T, and $0 \leq S_{max} \leq N-2$.

The following theorem gives the upper bound on the number of dual simplex pivots. Let u be the number of source nodes i in T^*, the optimal tree, such that $(0,i) \notin T^*$. Let S_{max} be the initial total

deficiency as defined before, and $q=|\{i : i\in V_F^0, S_i' \geq 0\}|$, that is q is the number of nodes i in T^0 with $S_i' \geq 0$.

Theorem 4.3. The total number of dual pivots in the algorithm for the assignment problem is bounded by

$$S_{max} + (u-1)(u-2)/2 - (q-1)(q-2)/2 \tag{4.2}$$

Proof: At the beginning of the algorithm, there can be at most (q-1) pivots before decreasing S_{max} by one. Thus after S_{max} decreases by one, we have at most q+1 nodes in V_F with $S_i' \geq 0$. Thus before the next reduction occurs in S_{max}, we can have at most q pivots. Continuing with this argument, when $S_{max} = 1$, we have at least two source nodes in T_p, thus we can have at most u-2 pivots before S_{max} reduces to zero. Summing up all the pivots, we get

$$S_{max} + \sum_{j=q-1}^{u-2} j = S_{max} + (u-1)(u-2)/2 - (q-1)(q-2)/2 \;. \blacksquare \tag{4.3}$$

Lemma 4.3. The worst case bound of the algorithm is (N-1)(N-2)/2 pivots.

Proof: Note that $S_{max} \leq N-2$, $q \geq 1$, and $u \leq N-1$. Thus by letting $S_{max} = N-2$, q=1 and u = N-1, we get

$$(N-2) + (N-3)(N-2)/2 = (N-1)(N-2)/2 \;. \blacksquare$$

4.4 Computational Experience

For assignment problems, we will compare the algorithm DUAL with three other algorithms. A primal simplex method, GNET, a primal-dual approach coded by Burkard and Derigs [44], and a dual simplex

We will call the primal-dual approach algorithm ASSIGN.

Test problems are randomly generated by the NETGEN program developed in [36]. For a fixed number of nodes, two sets of problems are generated; one being a sparse problem, designated by AS and the other being denser, designated by AD. Characteristics of the generated problems are given in Table 4.1.

Table 4.1 Characteristics of the Assignment Problems Generated

Problem Type	No. of Nodes	No. of Arcs	Cost Range
AS-1	100	250	1-100
AS-2	200	1000	1-100
AS-3	300	2250	1-100
AS-4	400	4000	1-100
AD-1	100	1250	1-100
AD-2	200	5000	1-100
AD-3	300	11250	1-100
AD-4	400	20000	1-100

The performance of the algorithms are summarized in Table 4.2, where the mean values are obtained by averaging ten runs. Note that for algorithm ASSIGN, the number of pivots criterion is not relevant since it does not perform simplex-like pivots.

The CPU times that are reported are in seconds, and they do not include the input-output time. The time statistics are collected using the library routines LIB$INIT_TIMER and LIB$STAT_TIMER.

Table 4.2 Performance of Algorithms for Assignment Problems

	No. of Pivots Min	Max	Mean	CPU Times Min	Max	Mean
Problem AS-1						
DUAL	39	73	57.0	0.10	0.26	0.18
DTRAN	92	117	104.6	0.24	0.43	0.31
ASSIGN	*	*	*	0.18	0.26	0.22
GNET	152	188	167.2	0.37	0.48	0.43
Problem AS-2						
DUAL	132	180	152.0	0.73	1.11	0.92
DTRAN	222	256	240.2	0.99	1.91	1.54
SSIGN	*	*	*	0.99	1.59	1.29
GNET	335	437	384.4	1.29	1.79	1.55
Problem AS-3						
DUAL	215	274	237.6	1.84	2.49	2.19
DTRAN	357	402	379.2	2.97	5.79	4.71
ASSIGN	*	*	*	2.94	4.55	3.24
GNET	523	726	637.6	2.85	3.95	3.45
Problem AS-4						
DUAL	296	354	337.8	3.58	4.68	3.97
DTRAN	502	561	529.5	10.71	13.75	11.91
ASSIGN	*	*	*	5.02	7.22	6.30
GNET	756	933	812.8	5.06	6.17	6.95

Table 4.2. (continued)

	No. of Pivots Min	Max	Mean	CPU Times Min	Max	Mean
Problem AD-1						
DUAL	54	80	69.0	0.21	0.50	0.36
DTRAN	103	124	114.0	0.48	0.90	0.70
ASSIGN	*	*	*	0.18	0.29	0.25
GNET	164	190	177.6	0.73	0.86	0.80
Problem AD-2						
DUAL	147	187	169.8	1.60	3.04	2.47
DTRAN	234	261	253.6	3.02	5.17	4.56
ASSIGN	*	*	*	1.28	1.82	1.49
GNET	392	547	448.6	3.85	5.80	4.51
Problem AD-3						
DUAL	226	255	243.0	4.87	7.12	5.60
DTRAN	393	436	418.5	15.18	21.69	17.27
ASSIGN	*	*	*	2.64	3.82	3.25
GNET	628	714	671.3	9.60	11.05	10.26
Problem AD-4						
DUAL	332	351	338.8	10.99	16.76	14.44
DTRAN	519	559	540.3	33.12	52.46	39.90
ASSIGN	*	*	*	5.85	9.83	7.89
GNET	900	1114	993.3	21.32	23.10	22.60

When we examine Table 4.2, we see that for sparse problems, algorithm DUAL is faster than all the other algorithms. It is 2-3 times faster than the other dual simplex algorithm DTRAN, and 1.5-2 times faster than GNET and ASSIGN.

However, for dense problems, algorithm ASSIGN is the fastest algorithm. It is 1.5-2 times faster than DUAL. But, DUAL is still 2-3 times faster than GNET and DTRAN.

CHAPTER 5
THE PROJECT SCHEDULING PROBLEM

Large complex projects with interdependent activities can be described by project networks. In a project network each arc represents an activity and each node represents an event signalling the completion of all activities leading into it. The structure of the network defines the relations between activities and events. A schedule associates an occurrence time with each event. Therefore, the project can be scheduled in several different ways. We assume that known amounts of cash flows are associated with each activity and with some events in the project. Hence given any feasible schedule the present value of all cash flows can be calculated. In this chapter, we consider a project scheduling problem introduced by Russel [45]. Given a project network and the cash flows associated with the activities and events, the problem is one of scheduling activities and events to maximize the present value of the outlays and receipts that occur during the execution of the project.

5.1 Formulation of the Problem

In a deterministic activity network G with arc set E and node set V, each arc $(i,j) \in E$ represents an activity and each node $k \in V$ represents an event signalling the completion of all activities leading into it.

Let $\beta(i)$ represent the "backward star" of node i which is all the activities leading into node $i \in V$, and let $\alpha(i)$ represent the "forward star" of node i which is all the activities emanating from node $i \in V$. There are two special nodes in V; node 1 and node n that represent the start and end of project, respectively. In this chapter, it is assumed that the activities are indivisible, i.e., once an activity starts it must be completed without any interruption. Moreover, we will assume that each node is on a directed path from node 1 to node n. Given a start time T_i of activity $(i,j) \in E$ there is an associated cash flow c_{ijt} to occur at time $T_i + t$, $t = 0, 1, \ldots, d_{ij}$ where d_{ij} is the duration of activity (i,j). If $c_{ijt} > 0$ then it is the revenue, and if $c_{ijt} < 0$, then it is the cost associated with the activity. Given the start time T_i of an activity (i,j) the net present value, NP_{ij} of (i,j) at time T_i is given by

$$NP_{ij} = \sum_{t=0}^{d_{ij}} c_{ijt}\, \alpha^{-t} \tag{5.1}$$

Here α^{-1} is called the one-period discount factor and given by $(1+i)^{-1}$ for discrete compounding and e^{-r} for continuous compounding, where i is called the interest rate and r is called the nominal interest rate.

Similarly, given the completion time T_j of an activity (i,j) the net future value of its cash flows at time T_j is given by

$$NF_{ij} = \sum_{t=0}^{d_{ij}} c_{ijt}\, \alpha^{(d_{ij}-t)} . \tag{5.2}$$

Note that for each activity $(i,j) \in E$, NF_{ij} and NP_{ij} agree in sign. In what follows we assume continuous compounding and thus $\alpha^{-1} = e^{-r}$.

Let T_i represent the realization time of event $i \in V$. Given such a feasible realization, for any $(i,j) \in E$ we have

$$T_i + d_{ij} \leq T_j \quad . \tag{5.3}$$

For an activity $(i,j) \in E$, if $NP_{ij} > 0$, then clearly (i,j) will be scheduled to start at time T_i, otherwise it will be scheduled to end at time T_j. Therefore regardless of the form of cash flows on the activities we can construct an equivalent cash flow problem in G where cash flows F_i occur only at T_i, $i \in V$, where

$$F_i = \sum_{\substack{(k,i) \in \beta(i) \\ NF_{ki} < 0}} NF_{ki} + \sum_{\substack{(i,j) \in \alpha(i) \\ NP_{ij} > 0}} NP_{ij} + \theta_i \qquad i \in V. \tag{5.4}$$

Here θ_i is a receipt associated with the realization of event $i \in V$.

Therefore, for given F_i associated with node i the problem becomes one of finding realization time T_i of each event i such that the present value of all F_i, $i \in V$ is maximized. It is assumed that there is an upper bound (R) on the duration of the entire project. In the case no such bound is available then without loss of generality R can be set equal to an arbitrary large positive number.

Let $T = (T_1, T_2, \ldots, T_n)$. The project scheduling problem may be formulated as :

(P1) $$\text{Maximize } f(T) = \sum_{i=1}^{n} f_i(T_i) = \sum_{i=1}^{n} F_i \exp(-rT_i) \tag{5.5}$$

Subject to

$$T_i - T_j \leq -d_{ij} \qquad (i,j) \in E \tag{5.6}$$

$$T_1 = 0 \tag{5.7}$$

$$T_n \leq R \tag{5.8}$$

where r is the nominal interest rate used for discounting. Without loss of generality, we have assumed that the project starts at time 0 and must be completed by time R. Note that $f_i(T_i)$ is a strictly increasing (decreasing) concave (convex) function for $F_i < 0$ ($F_i > 0$) for $T_i \geq 0$.

The statement of the project scheduling problem as such leads to a very interesting and critical conclusion that the usual "time critical path" may be irrelevant to the maximizing of present value; and that it may be the optimal policy never to finish the project. To illustrate the situation when the optimal policy is never to finish the project we refer to the case of "unbalanced" or "front loaded" bidding as described in [45].

A proposal for many types of contracted projects includes a list of jobs to be completed. The project contractor then bids a price for each job. The contract is awarded to the contractor whose total bid is the smallest. The contractor is then paid the price of each job as each job is completed. Given total project bid, it is to the contractor's advantage to increase the price for jobs to be completed early and decrease the price of those to be completed late in the project without changing the total bid. Not only does this strategy increase the present value of the project, but it may also provide an incentive for the contractor never to finish the project or delay it as long as possible.

However, if the contractee was to use the net present value of cash flows as a criterion such an occurrence could be avoided by imposing a sufficiently large penalty for delaying the project. We also note that this example illustrates a problem which is not uncommon

especially in highly inflationary economies. Potential applications of the model are almost as numerous as are the applications of critical path methods. Some specific applications are given in [45] as

(a) the pricing of delays in contractual situations,

(b) the optimum timing of logically dependent investment

(c) the avoidance of payment structures which positively discourage the early completion of a project,

(d) identifying "cost control"activities during the progress of a contract,

(e) the optimum timing of major projects in a developing national economy.

5.2 Literature Review

The project scheduling problem was introduced by Russel [45]. His approach for solving problem P1 is to linearize f(T) by taking Taylor series expansion of f(T) around a feasible solution T' and taking only the linear terms. The dual of the linearized problem possesses a minimum cost network flow structure. The dual problem is solved iteratively for each new approximation until the convergence to a local optimum is realized.

Grinold [46] proposed the following transformation of the problem:

$$(P2) \quad \text{Maximize} \sum_{i=1}^{n} y_i F_i \tag{5.9}$$

Subject to:

$$y_j K_{1j} \le 1 \quad , \quad (1,j) \in E \tag{5.10}$$

$$y_i K_{ij} - y_j \le 0 \quad , \quad i \ne 1,\ (i,j) \in E \tag{5.11}$$

$$-y_n \le -K_{n1} \tag{5.12}$$

where $K_{ij} = \exp(rd_{ij})$, $y_i = \exp(-rT_i)$. He further proved that P1 and P2 are equivalent. There is a one-to-one correspondence between the extreme points of P1 and P2, and for given T, if y(T) is optimal to P2 then T is optimal to P1.

Doersch and Patterson [47] solved the problem of maximizing the present value of cash flows subject to capital constraints.

5.3 The Project Scheduling Algorithm

The algorithm presented here is a modification of the algorithm developed by Erenguc and Tufekci [48].

5.3.1 The Extreme Point Property of Project Scheduling Problem [48]

In this section we characterize the extreme points of S and then present the theorem which establishes the basis for the proposed algorithm. For convenience we will formulate P1 as follows:

$$\text{Maximize } f(T) = \sum_{i=1}^{n} F_i\, e^{-rTi} \tag{5.13}$$

Subject to

$$T \in S \equiv \{ T: \ -T_j \le -d_{1j} \text{ for } (1,j) \in E;\ T_i - T_j \le -d_{ij} \text{ for } i \ne 1, (i,j) \in E;\ T_n \le R \} \tag{5.14-5.16}$$

Here we assume $d_{1j} > 0$, $(1,j) \in E$, and $F_1 = 0$; if not a dummy node 0 and a dummy activity $(0,1)$ with $d_{01} = \epsilon > 0$ and $F_0 = 0$ can be appended to G that will insure $T_j > 0$, $j=1,\ldots,n$. The optimal solution to the modified problem is given as $e^{-r\epsilon} f(T^*)$ where $f(T^*)$ is the optimal solution to P1. Thus, without loss of generality we assume $d_{1j} > 0$, for all $(1,j) \in E$. In what follows we assume $|E| = m$. We note that $T_1 = 0$ does not appear in this formulation. Using the matrix notation we have

$$\text{maximize } f(\mathbf{T}) \tag{5.17}$$

$$\text{s.t.} \quad \mathbf{AT} \leq d' \tag{5.18}$$

where $\mathbf{A}$ is an $(m+1)\mathrm{x}(n-1)$ constraint matrix, $\mathbf{T}$ is the column vector of variables $T_2,\ldots,T_n$, and $d' = [-\mathbf{d}, \ R\ ]^T$. Adding slack variables to (5.18) we get

$$\text{Maximize } f(\mathbf{T}) \tag{5.19}$$

$$\text{s.t.} \quad \mathbf{AT} + \mathbf{IS} = d' \tag{5.20}$$

We note that the matrix $\bar{\mathbf{A}} = [\mathbf{A},\mathbf{I}]$ has full row rank. Moreover, any basic feasible solution to (5.20) will have $T_2,\ldots,T_n$ basic since $T_i > 0, i=2,\ldots,n$. Let $\mathbf{B}$ be any feasible basis matrix of $\bar{\mathbf{A}}$. Also let the rows and the columns of $\mathbf{B}$ be arranged in such a way that the first $(n-1)$ columns and rows correspond to T_j, $j=2,\ldots,n$. Since $\mathbf{B}$ is an $(m+1)\mathrm{x}(m+1)$ matrix, there will be $m-n+2$ slack variables S_B in the basis. Thus $\mathbf{B}$ can be put in the form given below.

$$\mathbf{B} = \begin{array}{c|c|c|c} & T & S_B & \\ \hline T & P & 0 & n-1 \\ \hline S_B & Q & I & m-n+2 \\ \hline & n-1 & m-n+2 & \end{array}$$

We note here that each row of **P** corresponds to an arc (i,j) of the node-arc incidence matrix of the project network. Since **B** is a basis matrix, $\mathbf{B}^{-1}$ exists which in turn implies $\mathbf{P}^{-1}$ exists. From the theory of network flows [49], the matrix **P** induces a spanning tree. After rearranging and partioning the right hand side, the basic solution can be obtained by solving the following system in two stages.

$$\left[\begin{array}{c|c} P & 0 \\ \hline Q & I \end{array}\right] \begin{bmatrix} T \\ S_B \end{bmatrix} = \begin{bmatrix} d_T' \\ d_B' \end{bmatrix} \tag{5.21}$$

First by solving

$$\mathbf{PT} = \mathbf{d_T}' \tag{5.22}$$

as in dual variable computation of network simplex [49], and then using

$$(S_B)_{ij} = T_j - T_i - d_{ij} \tag{5.23}$$

for all those arcs which are not in the spanning tree. We have just shown that an extreme point solution to P1 can be characterized by a spanning tree of the project network. Naturally, for an extreme point solution to be feasible we must have $T_i > 0$, $i=2,\ldots,n$ and $S_{ij} \geq 0$ for all $(i,j) \in E$.

In the remainder of this section we will show that an optimal solution to the network scheduling problem occurs at an extreme point.

Grinold has shown that the optimal solution to P1 (P2) occurs at an extreme point of the polyhedral set S. He uses the equivalence of P1 and P2 to prove this result. In this section, we give an alternative proof for this property of P1 which unlike Grinold's proof does not depend on the equivalence of P1 and P2. Our proof is solely based on

the structure of P1. We chose to present this proof because it is in the spirit of and lays the groundwork for the algorithm to be presented in the next section.

Definition 5.1. Let $V' \subseteq V$ be a node set with the following properties. For each pair of nodes $i,j \in V'$ there exists at least one chain (arc directions ignored) P_{ij} between the nodes i and j such that for each arc $(x,y) \in P_{ij}$, $(x,y) \in E$ we have $T_x - T_y = -d_{xy}$, and for each arc $(w,z) \in E$, $w \in V'$ and $z \notin V'$ or $w \notin V'$ and $z \in V'$ we have $T_w - T_y < -d_{wz}$. We call the subnetwork G' induced by the node set V' and the arc set $E' = \{(i,j): (i,j) \in E,\ T_i - T_j = -d_{ij}\}$ a critically connected component of G.

Proposition 5.1. Given a critically connected component $G' = (V',E')$, there exists a tree with node set V' and arc set $E'' \subseteq E'$ which spans V' such that for each arc $(i,j) \in E''$, we have $T_i - T_j = -d_{ij}$.

Theorem 5.1. There exists a global optimum solution to P1 which is an extreme point of S.

Proof: Let $T \in S$ be an optimum solution to P1. Let $V_1, V_2, \ldots, V_k$, $k \le n$ be the critically connected components corresponding to this solution. We have $V_i \cap V_j = \phi$ for all $i \neq j$, and $\bigcup_{i=1}^{k} V_i = V$. If $k=1$, then $V_1 = V$ and thus there exists a spanning tree as described in Proposition 5.1. Thus choosing a spanning tree from V would characterize an extreme point solution.

If $k > 1$, then consider any component V_i. Define

$\alpha(V_i) = \{(x,y): x \notin V_i,\ y \in V_i,\ (x,y) \in E\}$

and

$\beta(V_i) = \{(x,y): x \in V_i,\ y \notin V_i,\ (x,y) \in E\}$

For each $(x,y) \in \alpha(V_i)$ we have $T_x - T_y < -d_{xy}$ for otherwise node x would have been in V_i. Similarly, for each $(x,y) \in \beta(V_i)$ we have $T_x - T_y < -d_{xy}$.

Let $P_i = \sum_{x \in V_i} F_x e^{-rTx}$. Now consider the following solution T':

$$T_x' = \begin{cases} T_x & \text{if } x \notin V_i \\ T_x + \Delta & \text{if } x \in V_i \end{cases} \tag{5.24}$$

where Δ is defined as

$$\Delta = \begin{cases} \min\limits_{(x,y) \in \beta(V_i)} \{s_{xy}\} = s_{pq} & \text{if } P_i \le 0 \\ -\min\limits_{(x,y) \in \alpha(V_i)} \{s_{xy}\} = -s_{pq} & \text{if } P_i > 0. \end{cases} \tag{5.25}$$

Note that minimization over a null set is assumed to yield $\Delta = 0$. Here s_{xy} is the slack on arc (x,y). Let the minimum occur at arc (p,q). The value of the objective function at T' is

$$f(T') = \sum_{x \notin V_i} F_x e^{-rTx} + \sum_{x \in V_i} F_x e^{-r(Tx+\Delta)} \tag{5.26}$$

$$= \sum_{x \notin V_i} F_x e^{-rTx} + e^{-r\Delta} P_i. \tag{5.27}$$

Evaluating $f(T') - f(T)$ we get

$$f(T') - f(T) = (e^{-r\Delta} - 1)\, P_i. \tag{5.28}$$

From the definition of Δ, we have $\Delta \geq 0$ for $P_i \leq 0$ and thus $(e^{-r\Delta} - 1)P_i \geq 0$. Similarly, we have $\Delta \leq 0$ for $P_i > 0$ and again $(e^{-r\Delta} - 1)P_i \geq 0$. Therefore T' is at least as good a solution as T. Moreover T' has at least one less critically connected component since at T' the component V_w where node q (or p) belongs is now critically connected to node set V_i. Continuing this argument we conclude that there exists a solution T^* which is at least as good as T and forms a single critically connected component. Thus any spanning tree of this component as described in Proposition 5.1 will define an extreme point.

Theorem 5.2. Every local maximum to (P1) is a global maximum.

Proof: See Grinold [46].

Theorems 5.1 and 5.2 present an interesting result in that although f(T) is not convex, the global maximum occurs at an extreme point of S and moreover, a local optimum is a global optimum.

5.3.2 Notation

Throughout Section 5.3 we will use the following notation.

A path in G is an alternating sequence $Z=v_0,e_1,\ldots,e_k,v_k$ of nodes and arcs such that $e_i = (v_{i-1},v_i)$ or $e_i = (v_i,v_{i-1})$. In the former case, e_i is called a forward arc of Z. In the latter case it is called a backward arc of Z.

Let $\Gamma=(V,E)$ be a spanning tree rooted at node 1. Define PR(x) to be the predecessor function indicating the node which precedes node x

on the unique path from node $x \in V$ to node 0 in Γ. Let $D(i)$ represent a depth function defined as the number of nodes traversed from node 0 to node i on a unique path in T.

Let V_F be the set of nodes $i \in V$ such that the arc $(PR(i),i)$ is a forward arc in Γ. Similarly, let V_B be the set of nodes $i \in V$ such that the arc $(PR(i),i)$ is a backward arc in Γ.

5.3.3 The Project Scheduling Algorithm, PUSH

In this section, we will present an algorithm which will solve (P1) optimally. Before giving the precise steps of the algorithm, we will describe the logic of the algorithm.

Let $\Gamma = (V,E)$ be any spanning tree of the project network rooted at node 1. We will call a subtree $\Gamma_i = (V_i,E_i)$, the subtree "induced" by node $i \in V_F$ if this subtree is obtained by dropping arc $(PR(i),i)$. We define P_i as the total present value of the cash flows occuring at nodes $j \in \Gamma_i$,

$$P_i = \sum_{j \in V_i} F_j e^{-rTj} \qquad (5.29)$$

where V_i is the node set of the subtree induced by node i. The logic of the algorithm is to delay all the events in the induced subtree Γ_i if the present value of cash flows P_i, in Γ_i, is negative.

Given an induced subtree $\Gamma_i = (V_i,E_i)$, we define a subtree $\Gamma_i' = (V_i',E_i')$ of Γ_i by the following procedure, PRUNE4.

PROCEDURE "PRUNE4"

INPUT : $\Gamma_i = (V_i, E_i)$, $D(k)$, $k \in V_i \cap V_F$.

OUTPUT : A subtree $\Gamma_i' = (V_i', E_i') \subseteq \Gamma_i$ with the property that for each $j \in V_{iF}$ $P_j \geq 0$, and its associated P_i' value.

BEGIN

INITIALIZE : Set $\Gamma_i' \leftarrow\!\!-\!\!-\!\!-\!\!- \Gamma_i$, $R = \{ j \neq i \in V_i \cap V_F : P_j < 0 \}$.

WHILE $R \neq \phi$ DO

Find $j \in R$ such that $D(j) \geq D(k)$, for all $k \neq j \in R$. Obtain $\Gamma_j = (V_j, E_j)$. Set $\Gamma_i' = (V_i' | V_j, E_i' | E_j \cup \{(PR(j), j)\})$.

Update P_j, $j \in V_i' \cap V_F$ and R.

ENDWHILE

Compute $$P_i' = \sum_{j \in V_i'} F_j e^{-rTj} \tag{5.30}$$

END

Here P_i' might be viewed as the individual contribution of subtree Γ_i' to the negative present value P_i. The algorithm advances the nodes in Γ_i if $P_i' < 0$, that is the individual contribution of Γ_i' is negative. If $P_i < 0$ but $P_i' > 0$, then the negativeness is caused by a set of nodes in an induced subtree Γ_k where $\Gamma_k \subset \Gamma_i$, and hence only the nodes in Γ_k should be advanced.

Let Q be the set of nodes with $P_i' < 0$. The algorithm starts with the longest path tree (early start tree) Γ^0, and finds a node $p \in Q$ with $P_p' < 0$ such that no other node j on the unique path from node p to node 1 will have $P_j' < 0$. Initially, an easy way of determining Γ_p ,$p \in Q$

is to check the depth functions of the nodes in Q and select Γ_p such that $D(p) = \min_{j \in Q} D(j)$.

Throughout the algorithm, after a particular pivot, k, Γ_p is determined while updating Γ^{k+1}. Let $W_{p'}$ be the set of nodes on the unique path from node p' found in pivot k to node 1 in Γ^{k+1}. Then new p is chosen to be the node $\alpha \in W_{p'} \cap Q$ with the smallest depth function. If $W_{p'} \cap Q = \phi$, then p is chosen to be the node $\alpha \in Q$ with the smallest depth function. If no such node p exists, then the algorithm terminates. Otherwise, the T_x values of the nodes $x \in V_p$ are advanced, i.e., the events are delayed, by $\Delta \geq 0$.

The quantity Δ is determined by the slacks of the out of tree arcs from node set V_p to node set $V \backslash V_p$. If no such arc exists, then Δ is set to $\Delta = R - T_n$ where R is the latest delivery time of the project. If there exists some arcs from node set V_p to node set $V \backslash V_p$, Δ is obtained from

$$\Delta = \min \{T_y - T_x - d_{xy} : x \in V_p,\ y \in V | V_p,\ (x,y) \in E\}. \tag{5.31}$$

The algorithm obtains the adjacent extreme point by dropping the arc (PR(p),p) out of Γ and adding the arc (n,1) if $\Delta = R - T_n$ or adding the arc (r,s) which defines Δ in equation (5.31), into the tree, Γ. We now give a formal statement of the algorithm.

ALGORITHM PUSH

INPUT: $G = (V,E)$, F_i, $i \in V$, d_{ij}, $(i,j) \in E$ and R, the project deadline, r, the nominal interest rate.

OUTPUT: Optimal timings T_i^* of events $i \in V$ and optimal cost $f(T^*)$.

PROCEDURE :

INITIALIZE: Find a longest path tree Γ^0 of G and let T^0 be the vector of path lengths from node 1 to all other nodes.

For each node $j \in V_F$ determine the subtree $\Gamma_j' = (V_j', E_j')$

compute P_j' by Procedure PRUNE4.

Define $Q = \{j : P_j' < 0,\ j \in V_F\}$. Set $p' = 0$.

MAIN STEP : WHILE $Q \neq \phi$ DO

BEGIN

IF $p' = 0$ THEN

Find a node $p \in Q$ such that $D(p) = \min_{j \in Q} \{D(j)\}$.

ELSE

Find a node $p \in Q \cap W_{p'}$ such that $D(p) = \min_{j \in Q \cap W_{p'}} \{D(j)\}$

ENDIF

Define $X = \{(i,j) : i \in V_p,\ j \notin V_p,\ (i,j) \in E\}$

IF $X = \phi$, THEN

$\Delta = R - T_n$.

Set $T_i = T_i + \Delta \qquad i \in V_p$

$T_i = T_i \qquad i \in V \backslash V_p$.

Update Γ by deleting arc $(PR(p), p)$ and adding arc $(n,1)$.

Update P_j', Q.

ELSE

Compute

$\Delta = \min \{ T_j - T_i - d_{ij} : (i,j) \in X\}$. Let the minimum occur at $(r,s) \in X$.

Set $T_i = T_i + \Delta$ $\quad i \in V_p$

$T_i = T_i$ $\quad i \in V \backslash V_p$.

Update Γ by deleting arc (PR(p),p) and adding arc (r,s).

Update P_j', Q.

ENDIF

IF $W_p \cap Q = \phi$, THEN

Set p'=0

ELSE

Set p'=p

ENDIF

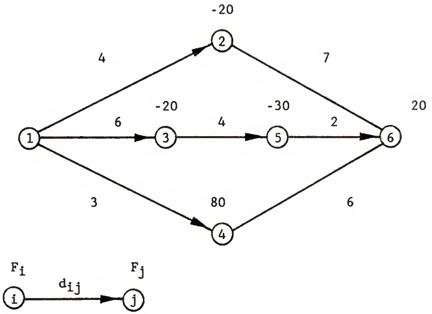
ENDWHILE

Compute f(T) and print T and f(T).

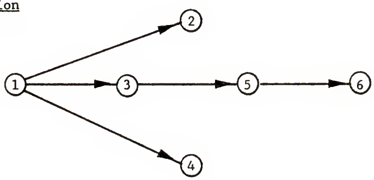
END.

We call each execution of the main step of this algorithm a "PUSH", since we are pushing the nodes of the subtree Γ_p always in one direction. We note that each execution of the main step corresponds to a simplex pivot in the sense that the algorithm moves to an adjacent extreme point in S as a result of the "PUSH" operation.

Example 5.1. The project scheduling problem with 6 nodes is solved in Figure 5.1. The nominal interest rate, r, is given as .10, and the project deadline, R, is given as 20.



Initialization



$T=(T_1,T_2,T_3,T_4,T_5,T_6) = (0,4,6,3,10,12)$

$P_2 = -20e^{-.1(4)} = -13.4064$

$P_3 = -60e-.1(6) - 30e^{-.1(10)} + 20e^{-.1(12)} = -37.9412$

$P_4 = 80\ e^{-.1(3)} = 59.2654$

$P_5 = -30e^{-.1(10)} + 20e^{-.1(12)} = -5.0125$

$P_6 = 20e^{-.1(12)} = 6.0239.$

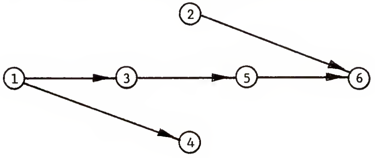
Figure 5.1 An Example

$P'_6 = P_6,\ P'_5 = P_5,\ P'_4 = P_4,\ P'_2 = P_2.$

$P'_3 = -60e^{-.1(6)} = -32.9287.$

$Q = \{ 2,3,5\}.$

Iteration 1. $p = 2$, arc (1,2) leaves, arc (2,6) enters, $\Delta = 1$.



$T = (0,5,6,3,10,12)$

$P_6 = -20e^{-.1(5)} + 20e^{-.1(12)} = -6.1067$

$P_5 = -20e^{-.1(5)} + 20e^{-.1(12)} - 30e^{-.1(10)} = -17.1431$

$P_3 = -20e^{-.1(5)} + 20e^{-.1(12)} - 30e^{-.1(10)} - 60e^{-.1(6)} = -50.0687.$

$P_4 = 59.2654.$

$P'_6 = P_6,\ P'_4 = P_4,$

$P'_5 = -11.0333$

$P'_3 = -32.9287.$

Iteration 2. $p=3$, arc (1,3) leaves. $X=\phi$, $\Delta = R-T_n = 20-12=8$, arc (6,1) enters.

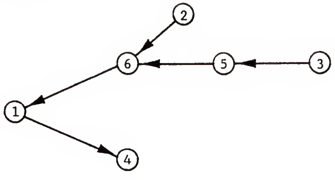


Figure 5.1 (continued)

$T = (0,13,14,3,18,20)$.

$P_4 = 59.2654$. $Q = \phi$. Current solution T is optimal.

Figure 5.1. (continued)

5.4. Computational Experience for Project Scheduling Problem

In this section, we present the computational experience obtained by the algorithm "PUSH" and compare it with Grinold's algorithm.

The difference between the algorithm "PUSH" and Grinold's algorithm is the selection of the leaving arc. In Grinold's algorithm an arc (x,y) is selected as a leaving arc if $w_{xy} < 0$. Here w_{ij} is computed by solving the following set of equations:

$$\sum_{j\in\beta_i} w_{ji} - \sum_{j\in\tau_i} w_{ij} = b_i \qquad \text{for } i=2,3,\ldots,n \tag{5.32}$$

where $b_i = F_i\exp^{(-rTi)}$, β_i is the set of events immediately preceding event i and τ_i is the events immediately following event i.

Since no selection rule is indicated in Grinold's paper, we assumed two different selection rules, one being the most negative w_{ij}, and the other being the first found negative w_{ij}. We coded two algorithms using these selection rules and named them GRINOLD1, which implements the first selection rule (most negative w_{ij}), and GRINOLD2, which implements the second selection rule (first encountered negative w_{ij}).

Algorithm "PUSH" chooses an arc (x,y) to leave the basis if (x,y) defines an induced subtree Γ_p' with $P_p' < 0$, and $D(p) = \min\{D(j), j\in Q\}$. Here Γ_p' does not contain any induced subtree $\Gamma_j \subseteq \Gamma_p$ with $P_j' < 0$. This

is equivalent to finding an arc (x,y) with a negative w_{xy} found by equations (5.32) after pruning negative w_{ij}'s computed earlier.

All of the algorithms tested are coded in FORTRAN 77 and are run on a VAX 11/750 using the VAX-11 Fortran V3.0 compiler. All runs are made overnight when we were the only users of the system.

We have tested the three algorithms for 12 sets of test problems for $n \in \{50,100,150,200\}$ and densities $\theta = \{0.1,0.3,0.5\}$, and used 10 replications in each set. Here $\theta = 2|E|/n(n-1)$ is the ratio of the number of actual arcs in the network to maximum possible number of arcs. In all problems a nominal interest rate of r=0.10 is used , and cash flows, F_i, are generated uniformly between (-10000,10000) for $i \neq n$ and F_n is set to 30,000.

The characteristics of the problems are summarized in Table 5.1. The performance of the algorithms are presented in Table 5.2 where the CPU times are in seconds, and the mean values are obtained from

Table 5.1. Characteristics of the Problems Generated

Problem Type	Number of Nodes	Number of Arcs
P-1	50	123
P-2	50	369
P-3	50	615
P-4	100	495
P-5	100	1485
P-6	100	2475
P-7	150	1118
P-8	150	3353
P-9	150	5590
P-10	200	1990
P-11	200	5970
P-12	200	9950

Table 5.2. Performance of the Algorithms

	No.of Pivots Min	Max	Mean	CPU times Min	Max	Mean
Problem P-1						
PUSH	21	29	25.4	0.20	0.38	0.31
GRINOLD1	22	36	26.2	1.10	1.77	1.29
GRINOLD2	44	100	60.8	1.79	3.91	2.51
Problem P-2						
PUSH	35	44	41.3	0.51	0.81	0.63
GRINOLD1	38	53	44.8	1.98	2.62	2.26
GRINOLD2	62	255	170.0	2.65	11.14	7.20
Problem P-3						
PUSH	35	49	43.8	0.62	0.98	0.80
GRINOLD1	39	54	47.1	1.76	3.05	2.46
GRINOLD2	84	312	194.3	3.74	13.52	8.51
Problem P-4						
PUSH	55	99	78.2	1.50	3.29	2.44
GRINOLD1	61	104	83.5	10.23	17.42	13.98
GRINOLD2	181	332	234.9	23.52	49.36	35.48
Problem P-5						
PUSH	82	109	97.4	2.90	4.73	4.00
GRINOLD1	94	123	109.0	16.68	21.22	19.15
GRINOLD2	399	782	678.0	64.47	123.89	103.54
Problem P-6						
PUSH	74	116	93.6	3.51	6.82	4.81
GRINOLD1	85	123	110.3	16.81	22.53	20.12
GRINOLD2	499	1484	936.8	78.20	218.79	120.82
Problem P-7						
PUSH	132	161	142.4	7.00	12.83	9.43
GRINOLD1	137	193	163.6	60.88	77.96	70.29
GRINOLD2	416	887	575.7	161.58	324.56	220.04

Table 5.2. (continued)

	No.of Pivots Min	Max	Mean	CPU times Min	Max	Mean
Problem P-8						
PUSH	125	174	151.2	9.41	14.92	12.28
GRINOLD1	167	228	187.3	69.49	94.86	78.00
GRINOLD2	384	1988	1371.6	340.20	420.86	509.10
Problem P-9						
PUSH	137	206	161.4	11.74	20.34	15.42
GRINOLD1	158	268	196.8	70.73	117.65	90.83
GRINOLD2	1729	4026	2701.0	664.31	1482.73	944.13
Problem P-10						
PUSH	129	229	175.6	9.24	25.06	17.23
GRINOLD1	205	285	226.3	147.60	201.25	175.37
GRINOLD2	488	1106	858.5	364.54	897.70	645.97
Problem P-11						
PUSH	168	243	197.7	21.23	34.98	23.39
GRINOLD1	216	297	266.1	160.38	218.80	205.28
GRINOLD2	1859	2527	2148.8	1204.86	1652.33	1395.35
Problem P-12						
PUSH	166	211	186.8	24.94	36.79	31.16
GRINOLD1	193	348	257.0	148.47	263.27	195.78
GRINOLD2	1025	5300	3880.1	705.94	4341.96	2374.40

When we examine Table 5.2, we see that algorithm PUSH is much faster than the other two algorithms. It is 3-10 times faster than GRINOLD1 and 8-76 times faster than GRINOLD2.

One of the reasons for having high CPU times for algorithms GRINOLD1 and GRINOLD2 is that in these algorithms all w_{ij}'s are recalculated after each pivot whereas in algorithm PUSH only those P_j'

values that are affected by the pivot are recalculated. Moreover, in algorithm GRINOLD1, for determining the leaving arc, all the w_{ij} values are checked to find the most negative one and this increases the computational time.

The reason for having larger number of pivots in algorithm GRINOLD1 is due to the selection rule of min $\{w_{ij}\}$. An arc (i,j) may have the most negative w_{ij} value, however, the negativeness can be caused because of other arcs. In our terms, P_j' may be positive but $P_j < 0$ due to the P_k' values of subtrees $\Gamma_k \subset \Gamma_j$ with $P_k < 0$. In such a case, the algorithm requires a "pulling back" pivot where events in Γ_j are advanced (not delayed). Algorithm GRINOLD2 requires no "pulling back" pivots, however the number of pivots required is enourmously large compared to the other two algorithms. The reason for this is that the algorithm can "push" separately several subtrees that are on a given path if each of these subtrees have negative P_k values, while algorithm PUSH can "push" all of these subtrees with a single pivot .

CHAPTER 6
SUMMARY

In this dissertation we focused on some special cases of optimal differential problems.

In Chapters 2-5, we developed three dual simplex algorithms for the minimum cost flow problem, and its special cases, the transportation problem and the assignment problem.

The leaving arc selection rule in the dual simplex algorithm, DUAL, provided us to find a computational bound for the algorithm of $O(M|V|^2)$ where M is the sum of the supplies (or demands) in the problem. By applying the scaling procedure developed by Edmonds and Karp [28], we developed the dual simplex algorithm,DUALSC which provided a polynomial computational bound of $O[(r+1)|V|^3]$. Since this bound is dependent on the logarithm of the input data of supply and demand values, DUALSC is not a genuinely polynomial algorithm. However, in terms of the magnitude of the bound, DUALSC provides the smallest value when we compare it with the bounds of other genuinely polynomial bounds of algorithms developed by Orlin [29] and polynomial bound of the algorithm DTRANSC, developed by Ikura and Nemhauser [32]. Orlin's two algorithms have computational bounds of $O(|V|^6\log|V|)$ and $O(|V^4|\log|V|)$. DTRANSC has a computational bound of $O[(r+1)|V|^5]$.

In our computational experimentation, we compared the algorithms DUAL and DUALSC with other dual simplex algorithms and the primal simplex algorithm, GNET. In previous experimentations reported in the literature, the primal simplex algorithms were found to be computationally superior to all other algorithms for large sized problems. In our experimentation, for minimum cost flow problems and transportation problems, algorithms DUAL and DUALSC performed better than all the other dual simplex algorithms for both small and large sized problems. They were 3-5 times faster than the algorithms developed by Orlin, and 1.5-2 times faster than DTRANSC. Even though algorithm DUAL was competitive with GNET for large dense problems, GNET performed better in large sparse problems by being 2-3 times faster than DUAL.

When algorithm DUAL is used to solve assignment problems, it becomes equivalent to Balinski's [43] dual simplex algorithm and provides the same bound of (N-1)(N-2)/2 pivots where $N=|V_S|=|V_D|$. However Balinski's algorithm cannot be extended to solve the minimum cost flow problem or the transportation problem.

In our computational experimentation for assignment problems, we concluded that algorithm DUAL is 2-3 times faster than other dual simplex algorithm, DTRAN and the primal algorithm GNET. When we compared DUAL's performance with a primal-dual algorithm developed by Burkard and Derigs [44], we concluded that DUAL is 1.5-2 times faster than the primal-dual algorithm for sparse problems but the primal-dual algorithm is 1.5-2 times faster than DUAL for dense problems.

There still remains an interesting open question as to whether there exists a simplex pivot rule that solves the minimum cost flow problem in polynomial time.

In Chapter 5, we developed an algorithm, PUSH, that solves the project scheduling problem introduced by Russel [45]. This project scheduling problem has an objective function of maximizing the present value of cash flows.

The difference between the algorithm PUSH and the algorithm developed by Grinold [46] to solve the same problem is the selection rule of the leaving arc in a given pivot. Since no selection rule is indicated in Grinold's algorithm, we assumed two different commonly used selection rules and compared these algorithms with algorithm PUSH in our computational experimentation. Based on our experimentation, we concluded that algorithm PUSH is 3-10 times faster than the algorithm which employs the leaving arc selection rule of the most negative value, and 8-76 times faster than the algorithm which employs the leaving arc selection rule of the first found negative value.

APPENDIX A
LIST OF NOTATIONS

Chapters 2-4:

P_i	: the unique path connecting node i to node 0 in T.
$T = (V, E_T)$	: a spanning tree rooted at node 0.
$T_i = (V_i, E_i)$	: a subtree for node $i \in V_F$ obtained by dropping arc (PR(i),i) out of T and taking the subtree that does not contain the root node.
$T_i' = (V_i', E_i') \subseteq T_i$	: a subtree for node $i \in V_F$ when all the subtrees $T_j' \subseteq T_i$ with positive S_j' values are pruned from T_i.
$T_i'' = (V_i'', E_i'')$	: a subtree for node $i \in V_B$ obtained by dropping arc (i,PR(i)) out of T and taking the subtree that does not contain the root node.
$T_i''' = (V_i''', E_i''') \subseteq T_i''$	: a subtree for node $i \in V_B$ when all the subtrees Tj' with $S_j' > 0$ are pruned from T_i''.
$\bar{T}_i$	: a subtree for node $i \in V_F$ when all the subtrees T_j''' with nonpositive S_j''' are pruned from T_i'.
Si	: negative of the flow on the forward arc (PR(i),i) for $i \in V_F$.
S_i'	: negative of the flow on the forward arc (PR(i),i) in T_i' for $i \in V_F$.
S_i''	: flow on the backward arc (i,PR(i)) for $i \in V_B$.

S_i'''	: flow on the backward arc $(i,PR(i))$ for $i \in V_B$ in T_i'''.
$\overline{S}_i$	: negative of the flow on the forward arc $(PR(i),i)$ in $\overline{T}_i$ for $i \in V_F$.
Z_i	: set of nodes j, with $S_j' > 0$ that are pruned from subtree T_i or T_i''.
Z_i'	: $Z_i \cup i$, if $S_i' > 0$; Z_i if $S_i' \le 0$.
$\overline{Z}_i$	: set of nodes j with $S_j''' \le 0$ that are pruned from subtree T_i'.
V_F	: set of nodes such that the arc $(PR(i),i)$ is a forward arc.
V_B	: set of nodes such that the arc $(PR(i),i)$ is a backward arc.
Q	: set of nodes j with $S_j' > 0$.
$K \subseteq Q$	: set of nodes j with $S_j' > 0$ and there exists no other $k \in P_j$ with $S_k' > 0$.
$(.)^b$	: a value(.) or a subtree (.) before a particular pivot.
$(.)^a$	: a value(.) or a subtree (.) after a particular pivot.
$(.)^z$	: a value(.) or a subtree (.) in the subproblem z.

REFERENCES

1. Rockafellar, R.T., Network Flows and Monotropic Optimization, John Wiley & Sons, Inc., New York, 1984.

2. Hitchcock, F.L., "The Distribution of a Product form Several Sources to Numerous Localities," Journal of Mathematics and Physics, 20, 1941, p.20.

3. Koopmans, T.C., "Optimum Utilization of the Transportation System," Proceedings of the International Statistical Conferences, Washington D.C., 5, 1949, p.136.

4. Dantzig, G.B., "Application of the Simplex Method to a Transportation Problem," in Activity Analysis of Production and Allocation, T.C. Koopmans, ed., John Wiley and Sons, New York, 1951.

5. Orden, A., "The Transshipment Problem," Management Science, 2, 1956, p.276.

6. Fulkerson, D.R., "An Out-of-Kilter Method for Solving Minimal Cost Flow Problems," J.Soc. Indust. Appl. Math., 9, 1961, p.18.

7. Ford, L.R., Fulkerson, D.R., "A Primal-Dual Algorithm for the Capacitated Hitchcock Problem," Naval Research Logistics Quarterly, 4, 1957, p.47.

8. Balas, E., Hammer, P.L., "The Dual Method for the Generalized Transportation Problem," Management Science, 12,1966,p.555.

9. Busacker, R.G., Gowen, P.J., "A Procedure for Determining a Family of Minimum-Cost Network Flow Patterns," ORO Technical Report 15, Operations Research Office, John Hopkins University, Baltimore, 1961.

10. Klein, M., "A Primal Method for Minimal Cost Flows with Applications to the Assignment and Transportation Problems," Management Science, 14, 1967, p.205.

11. Glover, F., Karney, D., Klingman, D., "Implementation and Computational Comparisons of Primal, Dual, and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems," Networks, 4, 1974, p.191.

12. Glover, F., Karney, D., Klingman, D., Napier, A., "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," Management Science, 20, 1974, p.793.

13. Glover, F., Klingman, D., "Improved Labeling of L.P. Bases in Networks," Research Report CCs 218, Center for Cybernetic Studies, The University of Texas, Austin, Texas, 1975.

14. Glover, F., Karney, D., Klingman, D., "The Augmented Predecessor Index Method for Locating Stepping-Stone Paths and Assigning Dual Prices in Distribution Problems," Transportation Science, 6, 1972, p.171.

15. Glover, F., Klingman, D., Stutz, J., "Augmented Threaded Index Method for Network Optimization," INFOR, 12, 1974, p.293.

16. Bradley, G.H., Brown, G.G., Graves, G.W., "Design and Implementation of Large Scale Primal Transshipment Algorithms," Management Science, 29, 1977, p.1.

17. Ali, A., Helgason, R.V., Kennington, J.L., Lall, H.S., "Primal Simplex Network Codes: State-of-the-Art Implementation Technology," Networks, 8, 1978,p.315.

18. Charnes, A., Glover, F., Klingman, D., Stutz, J., "Past, Present, and Future of Large Scale Transshipment Computer Codes and Applications," Computers and Operations Research, 2, 1975, p.71.

19. Cunningham, W.H., "A Network Simplex Method," Math Programming, 11, 1976, p.105.

20. Barr, R.S., Glover, F., Klingman, D., "The Alternating Basis Algorithm for Assignment Problems," Math Programming, 13, 1977, p.1.

21. Bland, R.G., "New Finite Pivoting Rules for the Simplex Method," Mathematics of Operations Research, 2, 1977, p.103.

22. Cunningham, W.H., "Theoretical Properties of the Network Simplex Method," Math of Operations Research, 4, 1979, p.196.

23. Srinivasan, V., Thompson, G.L., "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm," Journal of the Association of Computing Machines, 20, 1973, p.194.

24. Srinivasan, V., Thompson, G.L., "Accelerated Algorithms for Labeling and Relabeling of Trees with Application for Distribution Problems," Journal of the Association of Computing Machines, 19, 1972, p. 712.

25. Zadeh, N., "A Bad Network Problem for the Simplex Method and Other Minimum Cost Flow Algorithms," Mathematical Programming, 5, 1973, p.255.

26. Tomizawa, N., "On Some Techniques Useful for Solution of Transportation Problems," Networks, 1, 1971, p.173.

27. Bertsekas, D.P., "An Algorithm for the Hitchcock Transportation Problem", Proceedings of the 18th Allerton Conference on Communication, Control and Computing, Allerton Park, Iowa, 1979, p.962.

28. Edmonds, J.J., Karp, R.M., "Theoretical Improvements in Algorithms Efficiency for Network Flow Problems," J. Assoc. Comp. Mach., 19, 1972, p.248.

29. Orlin, J.B., "Genuinely Polynomial Simplex and Non-Simplex Algorithms for the Minimum Cost Flow Problem," Sloan Working Paper No. 1615-84, MIT, Cambridge, Massachusetts, December, 1984.

30. Rock, H., "Scaling Techniques for Minimal Cost Network Flows," in Discrete Structures and Algorithms, (U. Page, ed.) 1980, Carl Hanser, Munchen, p.181.

31. Armstrong, R.D., Klingman, D., Whitman, D., "Implementation and Analysis of a Variant of the Dual Method for the Capacitated Transshipment Problem," EJOR, 4, 1980, p.403.

32. Ikura, Y., Nemhauser, G.L., "A Polynomial Time Dual Simplex Algorithm for the Transportation Problem," Technical Report No. 602, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York, 1983.

33. Ikura, Y., Nemhauser, G.L., "Computational Experience with a Polynomial-Time Dual-Simplex Algorithm for the Transportation Problem," Technical Report No. 653, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York, 1984.

34. Tardos, E., "A Strongly Polynomial Minimum-Cost Circulation Algorithm," Combinatorica, 5, 1985, p.247.

35. Fujishige, S., An $O(m^3 \log m)$ Capacity Rounding Algorithm for the Minimum-Cost Circulation Problem : A Dual Framework of the Tardos Algorithm," University of Tsukuba, Japan.

36. Klingman, D., Napier, A., Stutz, J., "NETGEN- A Program for Generating Large Scale (Un)Capacitated Assignment, Transportation and Minimum Cost Flow Network Problems," Management Science, 20, 1974, p.814.

37. Kuhn, H.W., "The Hungarian Method for the Assignment Problem," Naval Research Logistics Quarterly, 2, 1955, p.83.

38. Hung, M.S., Rom, W.O., "Solving the Assignment Problem by Relaxation," Operation Research, 28, 1980, p.969.

39. Engquist, M., "A Successive Shortest Path Algorithm for the Assignment Problem," INFOR, 20, 1982, p.370.

40. Hung, M.S., "A Polynomial Simplex Method for the Assignment Problem," Operations Research, 31, 1983, p.595.

41. Bertsekas, D.P., "A New Algorithm for the Assignment Problem," Math Programming, 21, 1981, p.152.

42. Balinski, M., "Signature Methods for the Assignment Problem," Operations Research, 33, 1985, p.527.

43. Balinski, M., "A Competitive (Dual) Simplex Method for the Assignment Problem," Mathematical Programming, 34, 1986, p.125.

44. Burkard, R.E., Derigs, U., "Assignment and Matching Problems : Solution Methods with FORTRAN- Programs," Lecture Notes in Economics and Mathematical Systems, 8, 1980, p.1.

45. Russel, A.H., "Cash Flows in Networks," Management Science, 16, 1970, p.375.

46. Grinold, R.G., "The Payment Scheduling Problem," Naval Research Logistics Quarterly, 19, 1972, p.123.

47. Doersch, R.H., Patterson, J.H., "Scheduling a Project to Maximize its Present Value: A Zero-One Programming Approach," Management Science, 23, 1977, p. 882.

48. Erenguc, S.S., Tufekci, S., "A Greedy Algorithm for a Project Scheduling Problem," Research Report No. 84-4, Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida, February 1984.

49. Kennington, J.L., Helgason, R.V., Algorithms for Network Programming, John Wiley & Sons, Inc., New York, 1980.

BIOGRAPHICAL SKETCH

Canan A. Sepil was born September 25, 1958, in Izmir, Turkey. She received a bachelor's degree in August, 1980, and a master's degree in June, 1982 in industrial engineering from the Middle East Technical University in Ankara, Turkey.

Canan A. Sepil is a registered Professional Engineer in Turkey and is a member of Operations Research Society of America.

Canan A. Sepil is married to Mehmet Sepil and is a mother of one.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Süleyman Tüfekçi

Süleyman Tüfekçi, Chairman
Associate Professor of Industrial and Systems Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Horst W. Hamacher
Associate Professor of Industrial and Systems Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Jack D. Elzinga
Professor of Industrial and Systems Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Ralph Swain
Professor of Industrial and Systems Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Chung Y. Lee
Assistant Professor of Industrial and Systems Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Selçuk S. Erenguç
Associate Professor of Management and Administrative Services

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

December 1987

Hubert A. Bevis

Dean, College of Engineering

Dean, Graduate School